Universidade Federal
do Rio de Janeiro
Escola Politécnica

ANÁLISE DE DESEMPENHO DE REDES DE CACHE

Guilherme Thurler Borges

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

Orientador: Rosa Maria Meri Leão

Rio de Janeiro
Março de 2018

ANÁLISE DE DESEMPENHO DE REDES DE CACHE

Guilherme Thurler Borges

PROJETO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO.

Examinadores:

Profa. Rosa Maria Meri Leão, Dr.

Prof. Edmundo Albuquerque de Souza e Silva, Ph.D.

Prof. Guilherme de Melo Baptista Domingues, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2018

Thurler Borges, Guilherme

Análise de Desempenho de Redes de Cache/Guilherme Thurler Borges. – Rio de Janeiro: UFRJ/POLI – COPPE, 2018.

XII, 59 p.: il.; 29, 7cm.

Orientador: Rosa Maria Meri Leão

Projeto (graduação) – UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2018.

Referências Bibliográficas: p. 58 – 59.

1. Redes de cache. 2. Simulação. 3. Análise de desempenho. 4. Redes orientadas a conteúdo. I. Maria Meri Leão, Rosa. II. Universidade Federal do Rio de Janeiro, Escola Politécnica/ Curso de Engenharia de Computação e Informação. III. Título.

Dedico este trabalho aos meus pais e amigos que me apoiaram ao longo de minha formação.

Agradecimentos

Agradeço aos meus pais, por todo o suporte e carinho que recebi ao longo de toda a minha vida, por sempre me motivar a continuar com minha formação e estarem sempre presentes, tanto nos momentos bons como nos ruins.

Agradeço aos meus colegas e amigos, que me ajudaram imensamente em minha formação, seja com suporte e motivação ou ajudando-nos uns aos outros nas disciplinas da faculdade.

Agradeço ao professor Edmundo e à professora Rosa, professores do laboratório LAND, por me orientarem ao longo da vida acadêmica na faculdade, que agora culmina no êxito deste trabalho.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

ANÁLISE DE DESEMPENHO DE REDES DE CACHE

Guilherme Thurler Borges

Março/2018

Orientador: Rosa Maria Meri Leão

Curso: Engenharia de Computação e Informação

Distribuição de conteúdo é um tópico cada vez mais central na área de redes, em parte devido ao aumento de popularidade de plataformas de vídeo sob demanda e o surgimento de problemas de escalabilidade e desempenho atrelados a esse crescimento. Redes orientadas a conteúdo são uma das soluções propostas para resolver parte desses problemas, propondo uma alteração no modelo de internet atual, em que o nome do host e do cliente são as peças centrais da comunicação. Nesse novo paradigma, os conteúdos seriam o centro da rede: clientes passariam a buscar por conteúdos na rede ao invés de procurar em algum servidor específico. Essa mudança traz consigo uma série de problemas e desafios a serem resolvidos, alguns dos quais serão abordados ao longo desta monografia.

Utilizando um conjunto de dados reais fornecidos por uma empresa que possui uma plataforma de distribuição de vídeo sob demanda, avaliamos o desempenho da arquitetura de redes orientadas a conteúdo proposta em [1]. Foram consideradas diferentes políticas de inserção e remoção de conteúdos no cache assim como um algoritmo para busca do conteúdo na rede. Um simulador foi desenvolvido e as seguintes métricas foram calculadas: fração de requisições filtradas ou probabilidade de *hit*, tempo médio de vida no cache, número médio de escritas no cache. Experimentos foram executados considerando diversos cenários. Através dos resultados foi possível observar como os diversos parâmetros do sistema afetam o desempenho da rede.

Palavras-Chave: Redes de cache, Simulação, Análise de desempenho, Redes orientadas a conteúdo.

Abstract of the Undergraduate Project presented to Poli/COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

PERFORMANCE ANALYSIS OF CACHE NETWORKS

Guilherme Thurler Borges

March/2018

Advisor: Rosa Maria Meri Leão

Course: Computer and Information Engineering

Content distribution is a trending topic in the network research field, in part because of the growing popularity of on demand video platforms. A number of scalability and performance issues have arisen with the growth of video traffic. Content-oriented networks are one of the proposed solutions to solve some of these problems, proposing a change in the current Internet model, where the host and client are the central pieces of communication. In this new paradigm, however, the content names would take the central role: clients would now search for contents in the network, rather than search for a host that stores the content. This creates a new set of problems and challenges to be overcome, some of which are discussed in this monography.

Using a set of real data provided by a company that owns a on demand video distribution platform, we evaluated the performance of the content-oriented network architecture proposed in [1]. Different content insertion and removal policies into/from the cache were considered as well as an algorithm to search content on the network. A simulator was developed and the following metrics were calculated: fraction of filtered requests or hit probability, average lifetime in the cache, average number of writes in the cache. Experiments were performed considering several scenarios. Through the results it was possible to observe how the several parameters of the system affect the performance of the network.

Keywords: Cache networks, Simulation, Performance analysis, Content-oriented networks.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste capítulo serão apresentadas a visão geral sobre o tema abordado nesta monografia, além da sua contextualização, motivação, objetivo e contribuição.

1.1 Visão Geral e Contextualização

Faz parte do dia-a-dia de boa parte da população mundial acessar a internet e navegar por conteúdos de todo tipo, sejam fotos e vídeos ou artigos científicos e páginas pessoais. Junto ao crescimento do número de usuários da internet, vimos uma grande mudança estrutural da arquitetura dessa ao mudarmos de uma rede de comutação de circuitos para uma de comutação de pacotes [2]. A mudança viabilizou o uso da rede por várias pessoas ao mesmo tempo, ao deixar de dedicar canais de comunicação para cada conexão estabelecida para transmitir pedaços da informação pela rede, que trafegam independentemente pela rede para sair de algum servidor e chegar ao cliente.

Junto ao aumento do número de usuários, nota-se uma mudança do tipo de conteúdo acessado na internet, com a crescente popularidade de conteúdo em vídeo, seja em formato estático em que um arquivo é baixado por completo e então assistido, ou em formato de *streaming*, em que o arquivo é transmitido do servidor ao cliente enquanto este consome o que já foi recebido. É estimado que boa parte do tráfego de rede atual seja dessa natureza [2, 3], e que isso deveria implicar em uma mudança na arquitetura da rede da mesma forma que o crescimento exponencial de usuários impactou a mudança de comutação de circuitos para pacotes.

Nesse contexto surge a ideia de redes de cache, ou redes orientadas a conteúdo, em que o foco da rede deixa de ser quem disponibiliza conteúdos, e passa a ser os próprios conteúdos. A mudança proposta ainda está sendo muito estudada e desenvolvida [4], mas em sua essência há o fato de que usuários passariam a buscar por conteúdos na rede, e não em provedores, de maneira que isso não altere a forma como usamos a internet, ou seja, isso deveria ser feito de forma transparente ao

usuário final.

Como essa área de estudo ainda é muito recente, muitos problemas ainda estão em aberto e soluções propostas precisam ser testadas em diversas situações que variam de ambientes idealizados a ambientes reais, para avaliar sua viabilidade. Sendo assim, o objetivo deste trabalho é estudar uma solução proposta para resolver um dos problemas desse contexto [1, 3], utilizando dados reais fornecidos por uma empresa que possui uma plataforma de distribuição de vídeos sob demanda.

1.2 Motivação

Muitas soluções propostas para resolver problemas atrelados ao contexto de redes de cache ainda residem puramente no domínio analítico, sem qualquer experimento baseado em dados ou ambientes reais para validar a teoria na prática [4]. Um desses estudos [1, 3] aborda o problema de armazenar conteúdos de maneira distribuída na rede, e então buscar por esse conteúdo de maneira oportunística, utilizando algumas técnicas abordadas mais à frente neste trabalho.

Assim, é de interesse tanto dos autores das teorias propostas quanto do autor desta monografia que se tenha alguma ideia de como essa teoria se comportaria em um ambiente mais próximo do real, em particular simulando requisições de tráfego na rede baseadas em dados reais, sendo este então o objetivo principal deste trabalho.

1.3 Objetivo e Contribuição

O objetivo deste trabalho é a avaliação do desempenho da arquitetura proposta em [1, 3] usando dados de um sistema real. Será usado um log de requisições de conteúdos de uma plataforma de distribuição de vídeo para simular usuários realizando requisições em uma rede de cache. Diferentes políticas de inserção e remoção dos conteúdos serão consideradas assim como um algoritmo para busca dos conteúdos baseado em passeio aleatório.

Uma das principais contribuições desse trabalho é o desenvolvimento de um simulador orientado a eventos para avaliar o desempenho da arquitetura proposta. Além disso, definimos uma metodologia para análise dos dados do sistema real, que poderá ser aplicada em outras bases de dados. Ambas as implementações (o simulador e a metodologia para análise dos dados) estarão disponíveis em formato de código aberto, para que outros usuários possam usá-las em outros experimentos.

Por fim, destacamos como uma importante contribuição a realização de experimentos considerando diversos cenários, o que permitiu analisar como os diversos parâmetros do sistema afetam o desempenho da rede de cache.

1.4 Estruturação do Texto

Esta monografia está organizada da forma descrita a seguir. O Capítulo 2 apresenta um embasamento teórico, introduzindo conceitos, definições e terminologia que serão utilizados ao longo do texto. No Capítulo 3 realizamos um estudo sobre os dados reais que foram disponibilizados para este trabalho, mostrando os passos até chegarmos a um conjunto de dados que são mais fáceis de utilizar no simulador. O Capítulo 4 descreve as etapas para a construção do simulador de redes de cache, explicando cada parâmetro e cada métrica de interesse considerados. No Capítulo 5 mostramos os experimentos realizados com o simulador construído e os resultados observados. Finalmente, no Capítulo 6 apresentamos as considerações finais do estudo realizado, além de possíveis áreas a serem estudadas em trabalhos futuros.

Capítulo 2

Referencial Teórico

Neste capítulo, serão apresentados alguns conceitos teóricos sobre os temas principais abordados nesta monografia. Cada um deles se encontra um uma seção abaixo, com explicações relevantes ao que será exposto no restante do desenvolvimento do trabalho.

2.1 Redes de Cache

Atualmente existe uma grande demanda por distribuição de conteúdo na internet. É notavelmente crescente o uso pelos usuários de ferramentas de criação e distribuição de conteúdo, em particular no formato de vídeos e fotos. É estimado que este tipo de mídia constitua cerca de 80% do tráfego global no ano de 2017 [3]. A arquitetura empregada atualmente conta com a comutação de pacotes para prover esses conteúdos aos usuários, e funciona baseada em endereços de clientes e servidores: um cliente sabe o endereço do servidor que contém o conteúdo desejado e realiza uma requisição pelo mesmo, que é direcionada ao servidor, o qual então responderá ao endereço do cliente.

Hoje em dia há um problema de escalabilidade nessa arquitetura, em que o servidor que provê um conteúdo muito popular é sobrecarregado por requisições por este conteúdo. E em muitos casos, não se trata apenas de um único conteúdo, mas sim de um conjunto consideravelmente grande de conteúdos. Para tentar controlar a situação, muitas empresas distribuidoras de conteúdo utilizam-se de CDNs (*Content Distribution Networks*) para aliviar a carga em seus servidores. A ideia por trás disso é colocar o conteúdo requisitado fisicamente mais próximo do cliente, e com isso distribuir a carga de maneira tanto lógica como geográfica.

O conceito básico de redes de cache (ou em inglês, *cache networks*, ou ainda *information-centric networks*) é aliviar a carga de servidores de conteúdo utilizando-se da própria arquitetura da rede, sem a necessidade de construir uma CDN especificamente para esse propósito. Cada nó da rede passaria a armazenar conteúdos de

maneira probabilística, de forma a tentar distribuir a carga da rede [2], como podemos ver no esquema da Figura 2.1, que mostra a evolução da internet de comutação de circuitos para a comutação de pacotes, e então para a rede de caches.

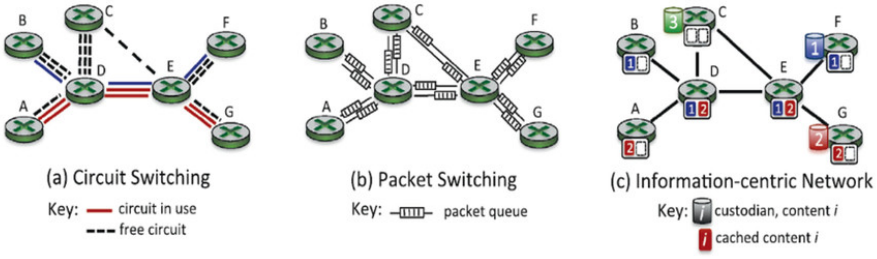

Figura 2.1: Comutação de circuitos, pacotes e conteúdo. Figura retirada de [2].

No exemplo da Figura 2.1, os nós D e E da rede conseguem servir os conteúdos 1 e 2, em azul e vermelho. Se por exemplo o nó A fizer uma requisição pelo conteúdo 1 (azul), ele pode recebê-lo diretamente de D ao invés de ter que receber de F. Porém, notamos que qualquer nó teria que buscar o conteúdo 3 (verde) em C, pelo fato deste não estar armazenado em nenhum cache da rede.

Há quatro grandes problemas a serem resolvidos para chegarmos a uma boa implementação dessa arquitetura. Todos os problemas estão sendo amplamente estudados e resultados antigos aprimorados [4]. Esses problemas a serem resolvidos incluem **como armazenar** conteúdos na rede, **onde armazenar** conteúdos na rede, **como buscar** conteúdos na rede e **como servir** conteúdos encontrados na rede [2].

Este trabalho terá seu foco na segunda e terceira questões propostas acima, onde armazenar e como buscar conteúdos. Sendo assim, é importante entender o porquê desses problemas existirem no contexto de redes de cache, e quais as abordagens possíveis de serem tomadas para tentar solucioná-los.

2.1.1 Armazenamento Oportunístico

Armazenar conteúdo nos nós intermediários da rede é um problema por causa da limitação de espaço que temos. Não podemos equipar todos os pontos de interesse da rede com sistemas de armazenamento capazes de guardar múltiplos conteúdos na ordem de *Gigabyte*, portanto temos que escolher bem quais conteúdos armazenar.

Isso é um problema mais notável ao tentarmos realizar *caching* de vídeos em alta definição, que facilmente consomem vários *Megabytes* de espaço em disco por minuto de filme. Porém devemos lembrar que o problema de *como* armazenar esses conteúdos não faz parte do problema a ser abordado neste projeto, mas sim *onde* armazena-lo. Sendo assim, devemos adotar uma política de armazenamento de conteúdo para controlar a entrada de conteúdos no cache.

Para facilitar a solução deste problema, assumiremos que a topologia da rede que estamos estudando segue um modelo hierárquico dividido em camadas, de forma que clientes que fazem requisições estariam na camada mais baixa e os conteúdos a serem buscados estão na camada mais alta. Esse modelo hierárquico não faz sentido ao analisarmos uma rede *peer-to-peer*, pois nela clientes também são provedores de conteúdo, e deveriam estar tanto na camada superior quanto na inferior, segundo a definição.

Consideramos que o processo de chegadas de requisições a um dado nó é um processo de Poisson com taxa λ e o interesse por um conteúdo decai com o tempo com taxa μ. Dessa forma, podemos definir políticas de armazenamento de conteúdos no cache, baseado nesses processos.

Uma abordagem seria utilizar um contador de requisições que seja decrementado periodicamente com uma taxa μ, e a decisão de incluir ou não um conteúdo no cache se daria pelo valor deste contador [3]. Outra abordagem seria atribuir ao conteúdo um tempo de vida no cache, e a cada nova requisição gerar um novo tempo, como se o interesse por aquele conteúdo fosse renovado com a nova requisição [5].

2.1.2 Busca Oportunística

Uma vez que uma requisição por um conteúdo é feita a partir de um cliente, a rede deverá buscar por algum nó intermediário que possua aquele conteúdo requisitado em cache, e caso não encontre em nenhum nó, buscar o conteúdo no servidor conforme seria feito hoje usando por exemplo uma CDN. Há diversas políticas para procurar o conteúdo pela rede, desde as mais simples como procurar somente nos nós que formam o caminho lógico do cliente ao servidor, até as mais complexas como buscar em todos os nós conhecidos da rede.

As duas políticas apresentadas acima não possuem desempenho muito bom, uma por não realizar uma busca na rede e portanto ter baixíssimas chances de encontrar o conteúdo antes do servidor, e a outra por realizar uma busca muito custosa na rede, tanto em termos de tempo quanto em termos de mensagens de controle na rede.

Aproveitando da topologia hierárquica proposta na Subseção 2.1.1, foi desenvolvido uma política de busca por conteúdos baseada em passeios aleatórios pelas camadas da rede [1, 3].

A Figura 2.2 ilustra como a requisição é tratada pela rede nessa política, onde as setas verdes representam os saltos entre camadas que formam o menor caminho entre o cliente e o servidor, as setas roxas representam o passeio aleatório pela camada atual, e a seta azul representa a transmissão do conteúdo encontrado para o cliente. Observamos que esta última se encontra no esquema somente para completar o

esquema, pois a maneira como o conteúdo é servido após ser encontrado na rede foge ao escopo deste trabalho.

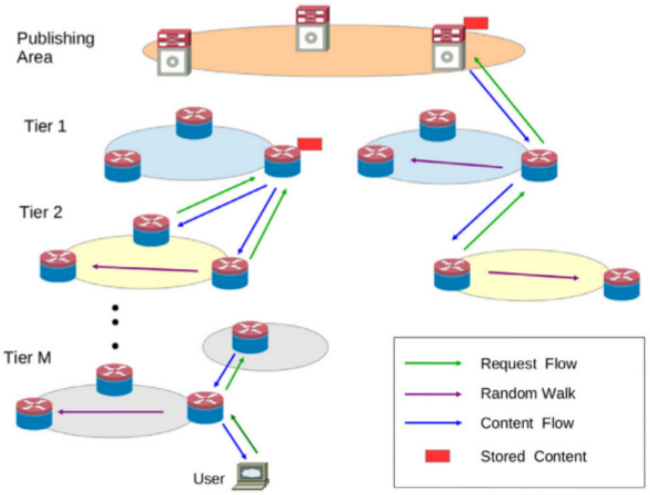

Figura 2.2: Esquema ilustrando a política de passeios aleatórios. Figura retirada de [3].

Notamos ainda na Figura 2.2 que a cada camada temos um único nó que recebe a requisição da camada inferior. A este nó damos o nome de nó de borda, aquele que faz parte do caminho entre o cliente e o servidor sem levar em conta os passeios aleatórios. A partir dele, um passeio aleatório com um número finito e limitado de passos é feito, e este é encerrado quando encontramos o conteúdo no passeio, ou o número máximo de saltos do passeio é alcançado.

Uma última observação sobre esse passeio aleatório é o fato do passeio não influenciar quem será o nó de borda da camada superior, pois este é determinado pelo nó de borda da camada atual. Sendo assim, podemos entender o passeio aleatório como uma ramificação do caminho principal a ser seguido, e não uma continuação ou pedaço dele.

2.2 Caching

Caching é uma técnica muito utilizada na área de redes desde os primórdios da internet [6], e consiste em armazenar alguma informação na rede de maneira que, ao buscar novamente por aquela informação, ela não precisará ser buscada em sua origem, mas sim onde fora armazenada.

A ideia de armazenar informação pela rede tem como objetivo principal reduzir a carga sobre o provedor daquela informação, distribuindo-a pela rede uma vez que nós intermediários conseguem servir aquele conteúdo como o servidor [1, 3].

Muitas questões surgem sobre como armazenar informações e conteúdos de naturezas distintas na rede, particularmente se referindo a conteúdos em vídeo, que

ocupam muito espaço em disco dependendo da qualidade do vídeo. Por exemplo, no caso de aplicações de vídeo, podemos armazenar apenas os primeiros 32MB de um arquivo de vídeo, possibilitando a um cache de 512MB armazenar 16 conteúdos distintos. Detalhes de implementação como esses, de compressão e fragmentação não fazem parte do escopo deste trabalho. Consideramos que todos os conteúdos a serem armazenados no cache terão um tamanho fixo, e portanto ocuparão o mesmo espaço. Sabendo quantos conteúdos podem ocupar o cache concorrentemente, o que resta a ser feito é definir uma política para gerenciar a inclusão e retirada de conteúdos no cache.

2.2.1 Políticas de Inserção e Remoção em Cache

Adotaremos duas políticas diferentes de inserção de conteúdos em cache ao longo desse trabalho. Uma delas baseada em um contador de requisições (RC) [1, 6], e outra baseada em um tempo aleatório de permanência no cache (TTL) [5, 6].

A política RC consiste em manter um contador de requisições que é incrementado toda vez que uma requisição por aquele conteúdo é feita ao cache, e ao atingir um limiar K, ele é inserido no cache. No caso trivial em que $K = 1$, o conteúdo é armazenado assim que uma requisição é feita.

Esse contador é decrementado a uma taxa μ, de forma que o contador seja decrementado μ vezes por unidade de tempo, em média. Esse processo é governado então por uma variável aleatória de distribuição exponencial com parâmetro μ.

Uma vez que o conteúdo atinge um limiar C, sendo $C < K$, ele é removido do cache. Damos nome de limiar de entrada do RC ao limiar K, e limiar de saída do RC ao limiar C. Na Figura 2.3 podemos ver a representação desse processo através de uma cadeia de Markov [6], em que os nós em vermelho representam que o conteúdo encontra-se disponível no cache. É importante notar que quando $K-C > 1$, dizemos que o contador possui histerese entre seus limiares de entrada e saída [1, 6].

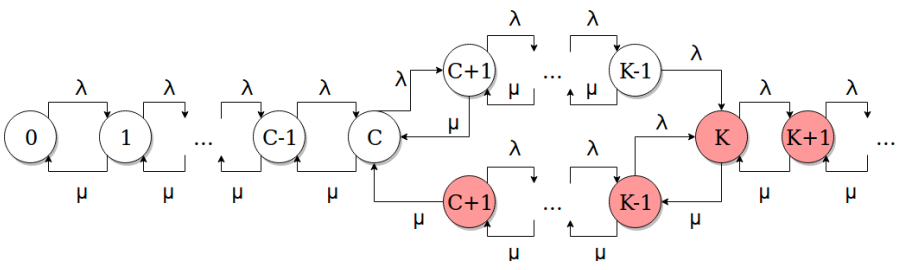

Figura 2.3: Cadeia de Markov representando a política de inserção RC.

A política TTL consiste em armazenar cada conteúdo no cache assim que uma requisição por ele é feita. É então associado a esse conteúdo um tempo de expiração, que será o tempo de vida daquele conteúdo no cache [5]. Se uma nova requisição che-

gar pelo mesmo conteúdo, caracterizando um cache hit, esse tempo será amostrado novamente e substituído pelo atual, efetivamente ”renovando”o tempo de expiração.

Embora a política se assemelhe muito à política RC com $K = 1$ e $C = 0$, nota-se que a política TTL não armazena o número de requisições feitas por aquele conteúdo, portanto a partir do momento que requisições pararem de chegar, esse conteúdo sairá do cache após o tempo de expiração, enquanto o valor do RC seria decrementado lentamente até o valor 0.

Na Figura 2.4 podemos ver uma cadeia de Markov que representa essa política. Notamos que a cadeia tem apenas 2 estados, ao contrário da que se refere ao RC, que possuía infinitos estados.

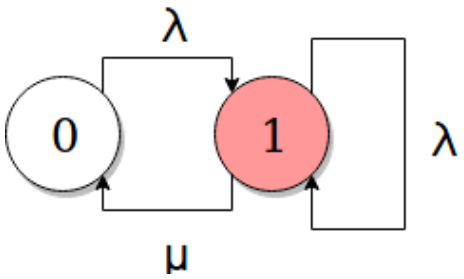

Figura 2.4: Cadeia de Markov representando a política de inserção TTL.

2.2.2 Políticas de Troca em Cache

Adotaremos quatro políticas diferentes de troca de conteúdos em cache ao longo desse trabalho. Uma delas baseada no contador de requisições (RC) [1, 6] descrito anteriormente, e as demais sendo políticas comumente usadas na literatura tratando-se de caching: FIFO, LRU e LFU.

A política de troca RC consiste em remover o conteúdo cujo valor de RC for o menor dentre os conteúdos armazenados no cache. Nota-se que mesmo que o valor do menor RC contido no cache seja maior do que o do novo conteúdo que será inserido, a troca continua acontecendo.

A política de troca FIFO consiste em remover o conteúdo que está há mais tempo no cache. Dessa forma, os conteúdos são removidos na mesma ordem em que são inseridos, assumindo que nenhum deles teve seu valor de RC decrementado abaixo do limiar de saída do RC (ou expirou no caso da política de inserção TTL).

A política de troca LRU consiste em remover o conteúdo cujo cache hit mais recente seja o mais antigo dentre os conteúdos do cache. Ou, em outras palavras, removeremos o conteúdo cuja requisição mais recente seja a mais antiga dentre os conteúdos do cache.

A política de troca LFU consiste em remover o conteúdo que tenha tido o menor número de cache hits ao longo da sua vida no cache. Essa política se assemelha muito

à política de maior RC, porém possui uma diferença: o valor de RC é decrementado periodicamente, ao passo que um contador de cache hits nunca será decrementado. Sendo assim, esta política é extremamente sensível a rajadas de requisições pelo mesmo conteúdo, pois o número de cache hits aumentará conforme o tamanho da rajada, e jamais será decrementado. É importante notar que, ao sair do cache, o contador de cache hits volta a ser zero.

Capítulo 3

Pré-Processamento dos Dados

Este capítulo apresenta os dados utilizados para estudar o comportamento de redes de cache, e como eles foram tratados e processados antes de serem usados no simulador. Cada seção tratará de cada etapa do estudo realizado sobre os dados, explicando o que foi feito para chegar ao objetivo de gerar um traço de eventos para o simulador.

3.1 Descrição dos Dados

Os dados fornecidos ao laboratório por uma empresa que possui uma plataforma de distribuição de conteúdo em vídeo *on demand* está organizado em um banco de dados SQLite [7]. Cada entrada no banco representa uma requisição à plataforma por algum conteúdo catalogado, que seria então enviado ao cliente para que este pudesse assistir ao conteúdo. Cada entrada no banco contém as seguintes informações úteis para este trabalho:

- ***Timestamp* da requisição**: Data em formato YYYY/MM/DD HH:MM da requisição pelo conteúdo
- **Título**: Título do conteúdo requisitado
- **ID do cliente**: Identificador numérico do cliente que fez a requisição
- **Região do cliente**: Nome da região em que o cliente se encontra
- **Cluster do cliente**: Nome do cluster que atende às requisições desse cliente

Notamos que o identificador numérico encontrado no banco deixa completamente anônimo o cliente que gerou aquela entrada no banco, mas o título, a região e o cluster estão formatados como *strings*, logo temos o nome de todos por extenso.

O *timestamp* da requisição possui informação da ordem de minutos, portanto não podemos distinguir a ordem em que as requisições com mesmo *timestamp* ocorreram.

Sendo assim, assumiremos que a ordem em que estão no banco é a mesma ordem em que ocorreram, para simplificar o processo de ordenar as requisições por *timestamp*.

Os campos *região* e *cluster* seguem uma lógica hierárquica. Um cluster está contido em uma região, ou uma região é dividida em vários clusters.

O conceito de *região* não está explícito na descrição da coluna do banco, portanto assumimos que se trata de alguma divisão do território nacional feita pela empresa para agrupar as requisições em grandes grupos de maneira geográfica. Existem três regiões diferentes no banco, denominadas "Leste", "São Paulo"e "Sul", reforçando essa ideia de tratar-se de uma divisão geográfica.

3.2 Estatísticas dos Dados

Primeiramente veremos como as requisições, os clientes e os conteúdos estão distribuídos pelas divisões geográficas observadas no banco de dados, a fim de verificar se estão uniformemente distribuídas ou se há algum favorecimento por alguma região ou cluster. Abaixo temos as estatísticas observadas para o banco todo:

- **Limites do timestamp**: De 01/12/2013 00:00 a 28/02/2014 23:59
- **Total de requisições**: 1004829
- **Total de clientes**: 25933
- **Total de conteúdos**: 5267

Na Tabela 3.1 vemos que a distribuição de requisições pelas regiões não é uniforme, mas o favorecimento da região leste sobre a região sul é pequeno, pois uma parcela considerável das requisições ainda vêm da região sul. O mesmo pode ser dito para a distribuição de clientes, indicando que as duas estatísticas podem estar correlacionadas: a região com mais clientes terá a maior quantidade de requisições. A porcentagem de conteúdos nos indica que, de todos os conteúdos disponíveis no catálogo, 90% foram requisitados pelos clientes da região leste, e similarmente para as demais regiões. Isso nos mostra que uma parcela pequena dos conteúdos disponíveis se quer é requisitado em alguma região.

Nome	Requisições	Clientes	Conteúdos
Regional Leste	43%	42%	90%
Regional São Paulo	34%	30%	86%
Regional Sul	23%	28%	80%

Tabela 3.1: Estatísticas do banco para cada região.

Na Tabela 3.2 podemos observar as mesmas estatísticas para cada cluster de cada região. As linhas horizontais da tabela separam os clusters por região. Notamos que

há sempre um cluster que predomina sobre os outros da mesma região, sendo a região sul a única que poderia ser chamada de homogênea ao observarmos a distribuição de requisições. Sendo assim, agrupar as requisições por cluster não terá resultados muito positivos se o objetivo for distribuir a carga na rede.

Nome	Requisições	Clientes	Conteúdos
Cluster Nordeste	18%	17%	76%
Cluster Salvador	6%	6%	61%
Cluster Rio de Janeiro	6%	6%	55%
Cluster Minas Gerais	5%	6%	53%
Cluster Vitória e Campos	4%	5%	50%
Cluster Belém	4%	4%	49%
Cluster São Paulo	31%	28%	85%
Cluster Grande Campinas	3%	3%	43%
Cluster Vale do Itajaí	13%	15%	70%
Cluster Curitiba	10%	10%	63%

Tabela 3.2: Estatísticas do banco para cada cluster.

Atentando agora para a taxa de requisições por minuto que observamos no banco de dados, vemos pela Figura 3.1 que há alguns minutos em que temos um pico de acessos, com valores variando de 25 a 40, mas que na maior parte do tempo, temos poucas requisições sendo geradas, inclusive com uma parcela considerável tendo zero requisições por minuto. Isso pode indicar que há períodos de maior ociosidade no sistema, em que poucas requisições são geradas por minuto.

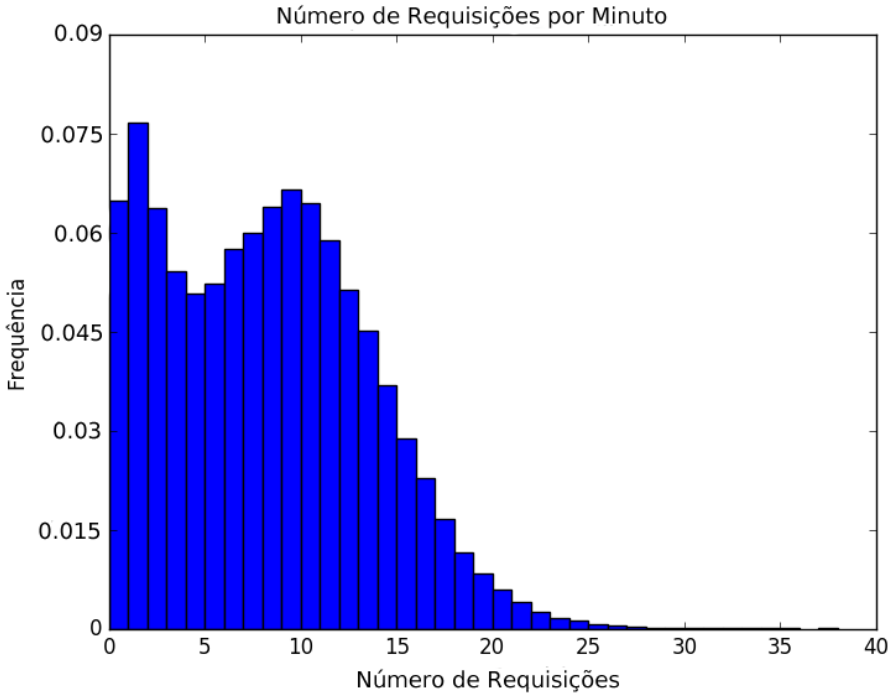

Figura 3.1: Histograma do número de requisições recebidas por minuto.

Uma última estatística a ser analisada é o número de requisições por conteúdo. Para cada conteúdo único no banco (identificado pelo seu título), contamos quantas requisições são feitas pelo mesmo ao longo de todos os registros, e vemos na Figura 3.2 essa distribuição. Atentamos para o fato do gráfico estar em escala log em ambos os eixos, o que nos mostra que uma fração muito pequena dos conteúdos possui um número muito alto de requisições, ao passo que a maior parte dos conteúdos possui um número muito baixo de requisições. Isso nos leva a crer que há uma forte disparidade entre a popularidade dos conteúdos do catálogo.

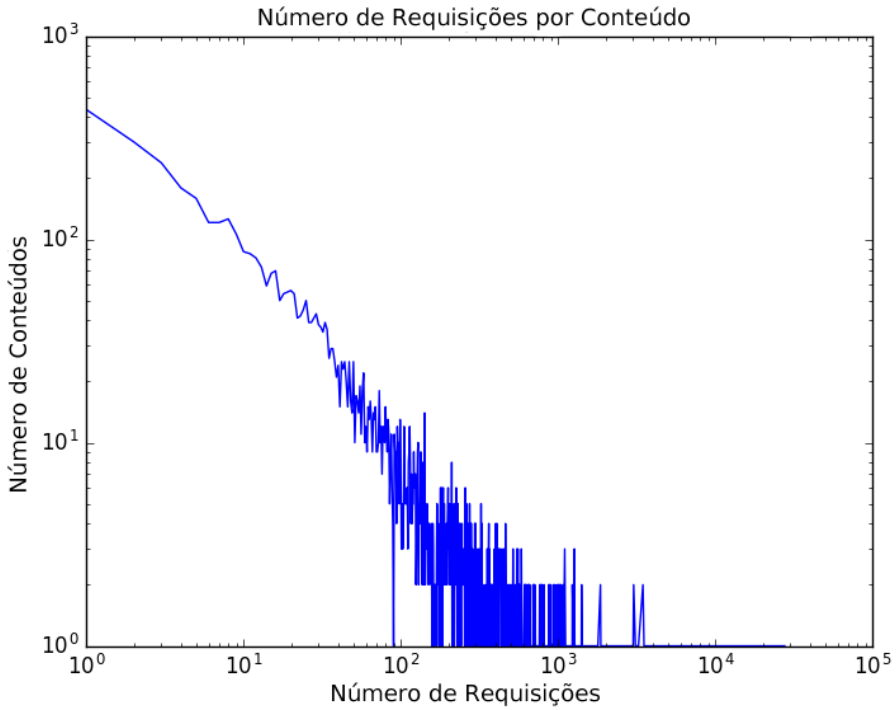

Figura 3.2: Histograma do número de requisições recebidas por conteúdo.

3.3 Análise em Séries Temporais

Uma abordagem para estudarmos o comportamento de cada conteúdo e seu conjunto de requisições é transformando esse conjunto de requisições em uma série temporal. Para tanto, consideremos um conteúdo C que possui N requisições registradas no banco de dados, cada uma feita em algum tempo t. Teremos então um array de tempos T descrito da seguinte forma:

$$T_C = [t_1, t_2, t_3, \ldots, t_N]$$

Outra maneira de visualizar esse dado é através da diferença entre cada *timestamp* de cada requisição, ou seja, o intervalo de tempo entre duas requisições. Dessa forma, se $N > 1$, conseguimos definir nossa série temporal S da seguinte maneira:

$$S_C = [t_2 - t_1, t_3 - t_2, \ldots, t_N - t_{N-1}]$$

Com essa abordagem, acabamos excluindo os conteúdos que tiveram uma única requisição ao longo do período amostrado. Como o foco deste trabalho é verificar o desempenho de redes de cache que tenham um espaço em cache finito, é importante não descartar esses conteúdos, pois dependendo da configuração da rede eles seriam armazenados no cache. Dessa forma, definiremos o tempo inicial do banco de dados como uma amostra para esses conteúdos, de forma que, se $N = 1$, teremos que:

$$S_C = [t_1]$$

A ideia de usar as diferenças entre amostras de *timestamps* é a de aproveitar que diferenças pequenas terão maior chance de representarem um cache hit para aquele conteúdo, pois se após a primeira requisição o conteúdo entrar no cache e logo após chegar outra requisição, o cache terá cumprido seu papel. A informação de quanto tempo passará entre requisições é muito importante para parametrizarmos os caches corretamente, e com isso melhorar seu desempenho. Usando essa abordagem, teremos uma série temporal para cada conteúdo catalogado na base de dados.

Para parametrizar o cache, precisamos definir a a taxa média de requisições por unidade de tempo, que chamaremos de λ. Como temos diversos conteúdos, teríamos um valor λ_C para cada conteúdo C, e teríamos duas abordagens para parametrizar nosso cache: cada conteúdo terá seu conjunto de parâmetros individual, ou teremos um único conjunto de parâmetros que buscará se aproximar ao máximo da média (ou alguma outra estatística) do nosso conjunto de valores de λ_C.

Calcular um conjunto de parâmetros para cada conteúdo C é possível, mas adiciona muita informação ao cache pois este conjunto crescerá linearmente com o número de conteúdos disponíveis. À medida que novos conteúdos forem criados, ou se tornarem obsoletos, toda essa informação deverá continuar armazenada no cache, pois ele terá uma política para cada conteúdo. O gerenciamento desses parâmetros e o grande número de conteúdos a serem tratados tornam essa abordagem pouco atrativa.

Se ao invés disso tomarmos um único conjunto de parâmetros para todos os conteúdos, sacrificaremos o desempenho do cache pela simplicidade dos parâmetros, que favorecerão um tipo específico de conteúdo.

Para resolver esse problema, buscamos um meio termo, em que teríamos classes de conteúdos que seriam parametrizadas individualmente em cada cache, e cada conteúdo estaria associado a uma classe, que seria por sua vez parametrizada de acordo com os conteúdos que fazem parte dela.

3.4 Clusterização de Séries Temporais

Intuitivamente, podemos clusterizar nossos conteúdos de duas formas: de acordo com o número de requisições pelo conteúdo ao longo do período observado, ou de acordo com os intervalos entre requisições pelo conteúdo. Na primeira abordagem, buscamos separar conteúdos que recebem muitas requisições dos que recebem poucas. Na segunda abordagem, separamos conteúdos que recebem requisições em rajada dos que recebem requisições espaçadas. Optamos pela segunda abordagem pois para avaliar políticas de cache, é importante agrupar conteúdos que tenham padrões de rajada *semelhantes*.

O problema de clusterizar séries temporais ainda é muito estudado. Possui diversas abordagens [8], e diversos algoritmos estão implementados em pacotes de suporte a algumas linguagens de programação muito utilizadas em análises estatísticas [9].

O maior problema a ser considerado na clusterização de séries temporais é definir um conceito numérico de similaridade entre duas séries [9], de forma que se possa então comparar essa similaridade em algoritmos de clusterização para chegar a uma divisão das suas amostras. Essa métrica de similaridade por vezes é chamada de distância na literatura, e o problema então é definir uma função f que atribui um valor numérico x à essa distância entre duas séries A e B:

$$f = distancia(A, B); f(A, B) = f(B, A) = x$$

Como não sabemos a distribuição de probabilidade que deu origem às nossas amostras em cada série S_C, devemos usar uma métrica de distância que seja *model-free*, ou seja, que é agnóstica à distribuição que deu origem às amostras. Poderíamos tentar encaixar uma distribuição exponencial com algum parâmetro λ_C a cada uma de nossas séries, porém como muitos conteúdos possuem poucas amostas para nos guiar, seria uma aproximação extremamente grosseira.

Sendo assim, usaremos a distância DTW (do inglês *Dynamic Time Warping*) para calcular uma distância para cada par de séries temporais que temos. A distância DTW busca encontrar um mapeamento r dentre todos os mapeamentos possíveis R em que cada amostra de A é associada a uma amostra de B, respeitando a ordem das amostras em cada série, de maneira a minimizar a soma da distância entre cada par de amostras [9]:

$$d_{DTW}(A, B) = \min_{r \in R} \left(\sum_{i=1}^{N} |A_i - B_i| \right)$$

A opção pelo DTW vem do fato de que buscamos encontrar padrões de acesso similares entre as séries, ao invés de buscar uma semelhança numérica ao longo de toda a série. Por exemplo, dois conteúdos que possuam uma rajada de requisições muito

parecida seguida por um longo período sem requisições devem ter uma distância entre si menor do que com um terceiro conteúdo que tenha um padrão de acesso mais regular, por mais que as rajadas ocorram em momentos diferentes das séries.

Uma vez que calculamos essa distância para cada par de séries temporais, podemos utilizar um algoritmo de clusterização para separar os conteúdos em classes. Utilizamos um algoritmo de clusterização hierárquica por não sabermos ao certo em quantas classes buscamos separar nossos conteúdos, e através do dendrograma gerado pelo algoritmo podemos facilmente determinar quantos e quais clusters adotaremos como uma solução para nosso problema.

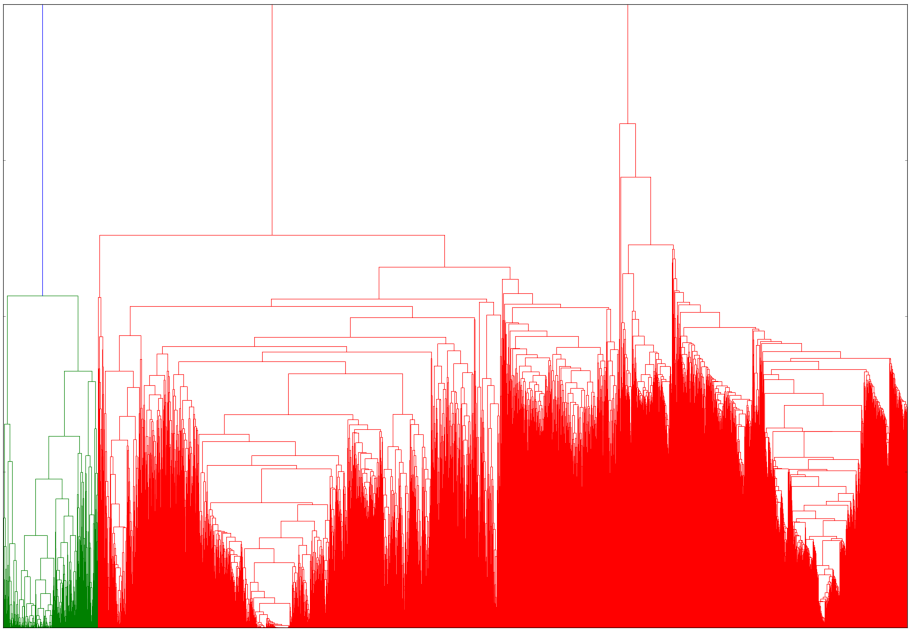

Figura 3.3: Dendrograma resultante do algoritmo de clusterização hierárquica.

Na Figura 3.3 vemos que o dendrograma apresentado não está completo, faltam duas conexões no topo da figura: a junção entre os clusters do centro e da direita (ambos em vermelho), e depois o do cluster da esquerda (em verde) com os demais. O motivo para eles não aparecerem na imagem é por conta da escala, pois ao incluir a primeira junção reduziríamos a escala da figura em 10 vezes, e se incluíssemos ambas as junções, em 40 vezes, o que deixaria difícil de ver o restante da figura.

Sendo assim, por conta dessa diferença de escala, fica claro que há uma diferença muito forte entre o cluster verde e os demais clusters em vermelho, o que faz sentido ao analisarmos quem são os conteúdos verdes: são os conteúdos que possuem o maior número de requisições e cujas séries são longas e formadas principalmente por valores pequenos, muitas vezes caracterizando uma rajada de acessos.

A separação entre os dois clusters em vermelho separam os conteúdos que são mais frequentemente acessados dos que quase nunca são acessados. Uma observação a ser feita é a de que ”frequentemente”não se refere ao número absoluto de acessos,

mas sim ao tempo entre acessos. Ou seja, em um cluster temos conteúdos cujas requisições são menos espaçadas do que no outro cluster, por mais que o número de acessos total seja menor.

É muito importante notar que essa separação não é ótima, e que muitos conteúdos poderiam estar melhor classificados em outro grupo, se essa classificação fosse feita à mão. Porém lembramos que seria inviável manualmente classificar todos os conteúdos, e que mesmo assim teríamos algum erro com um número de classes pequeno.

Como paramos as divisões em clusters ao atingirmos três clusters, classificaremos cada classe da seguinte maneira, daqui em diante:

- Conteúdos muito populares (ou de popularidade alta)
- Conteúdos populares (ou de popularidade média)
- Conteúdos pouco populares (ou de popularidade baixa)

Podemos ver na Figura 3.4 o boxplot do intervalo entre requisições de uma amostra de conteúdos caracterizados como muito populares, e na Figura 3.5 o boxplot para conteúdos caracterizados como pouco populares. A média está representada pelo quadrado vermelho, e a mediana por um traço vermelho. Podemos ver que conteúdos muito populares têm uma variância consideravelmente menor nas suas amostras, além de uma média e mediana bem mais baixas se comparadas ao dos classificados como pouco populares, o que nos indica que a separação foi, de certa forma, bem sucedida.

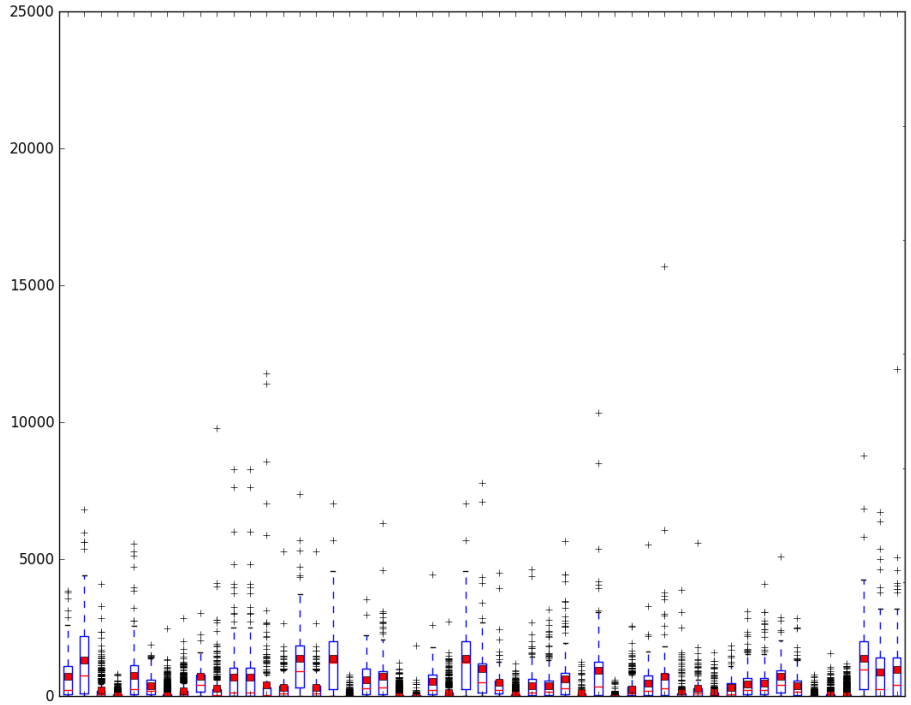

Figura 3.4: Boxplot das amostras de alguns conteúdos muito populares.

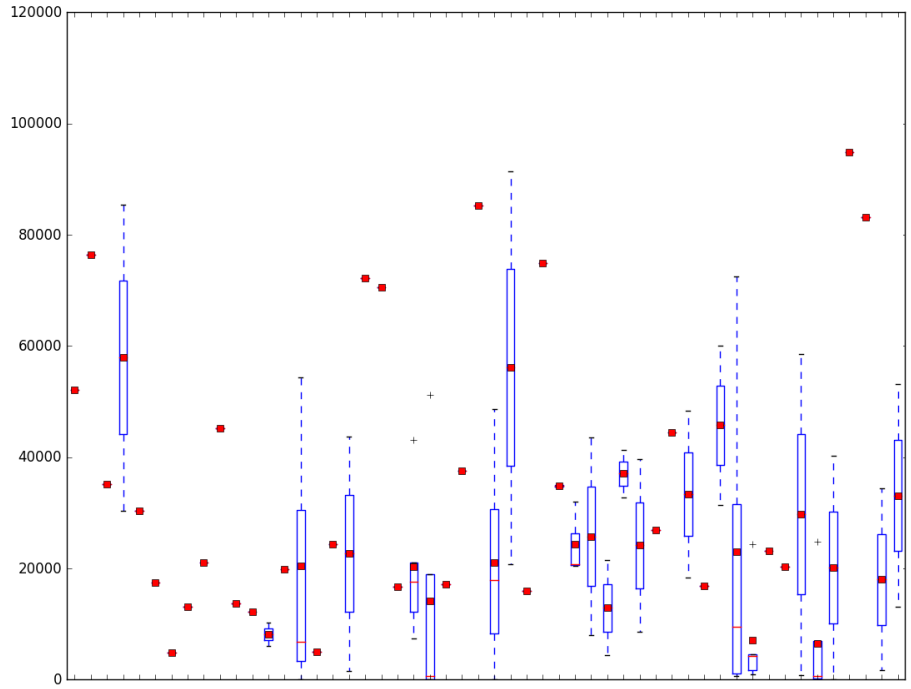

Figura 3.5: Boxplot das amostras de alguns conteúdos pouco populares.

3.5 Extração de Parâmetros para cada Classe

Feita a separação dos conteúdos em classes, devemos extrair um valor de λ para cada cluster, para que assim possamos parametrizar o cache. Utilizaremos então alguma estatística calculada a partir do λ_C de cada conteúdo de cada classe. Notamos que esse procedimento não é exato por natureza, ou seja, a escolha do parâmetro

final a partir dos dados mostrados tem um certo grau de subjetividade atrelado a ele. Porém, como esse parâmetro escolhido nunca representará fielmente a classe por esta ser bem heterogênea, não deve haver muita diferença nos resultados deste estudo se ele fosse um pouco maior ou menor, o que importa mais é a ordem de grandeza do valor escolhido.

Tendo isso em vista, observamos o histograma de medianas do intervalo entre requisições para conteúdos muito populares na Figura 3.6. Notamos um pico na frequência de medianas, portanto buscaremos algum valor nesta região. Observando os bins e as frequências, chegamos à conclusão de que 40 seria um bom valor para representar o valor $1/\lambda$ para a classe mais popular. Ressaltamos novamente que a precisão desse valor é de pouca importância - a ordem de grandeza será o fator decisivo no impacto deste parâmetro na simulação.

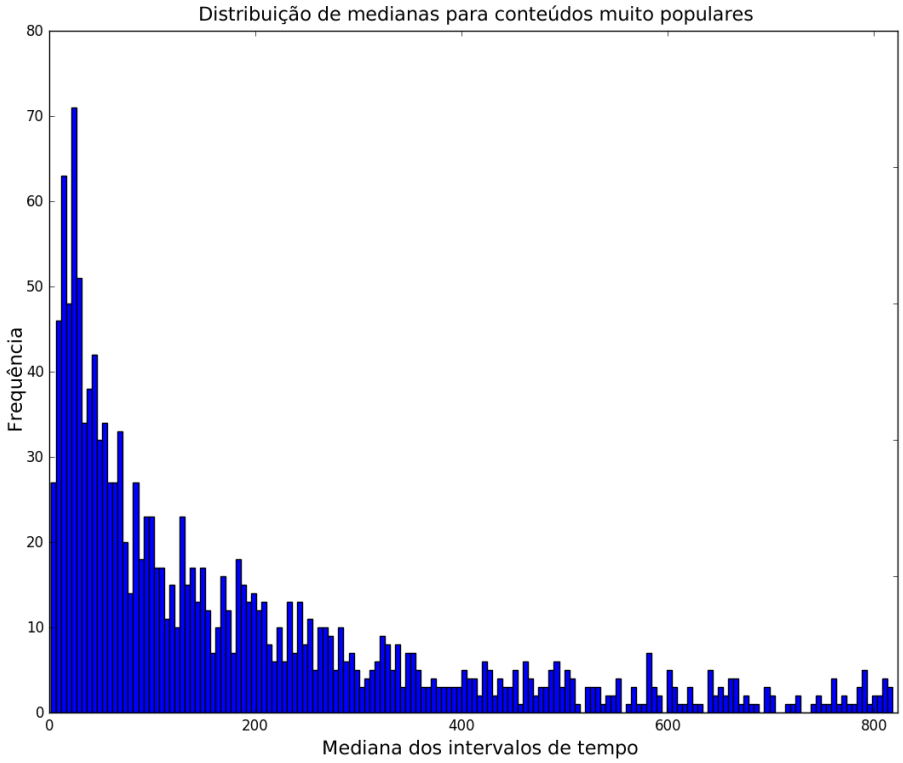

Figura 3.6: Histograma de medianas para conteúdos muito populares.

Para finalizar nossa análise dos parâmetros, temos abaixo na Figura 3.7 um q-q plot comparando um dos conteúdos muito populares com amostras de uma variável aleatória exponencial parametrizada com $\lambda = 1/40$. Notamos que há uma forte correlação entre as amostras, exceto pela cauda da distribuição, o que já era de se esperar devido ao número reduzido de amostras e ao fato da distribuição ter cauda pesada. Notamos que para outros conteúdos da mesma classe esse q-q plot não mostrou essa forte correlação, mas ela esteve presente em diversos conteúdos amostrados da classe.

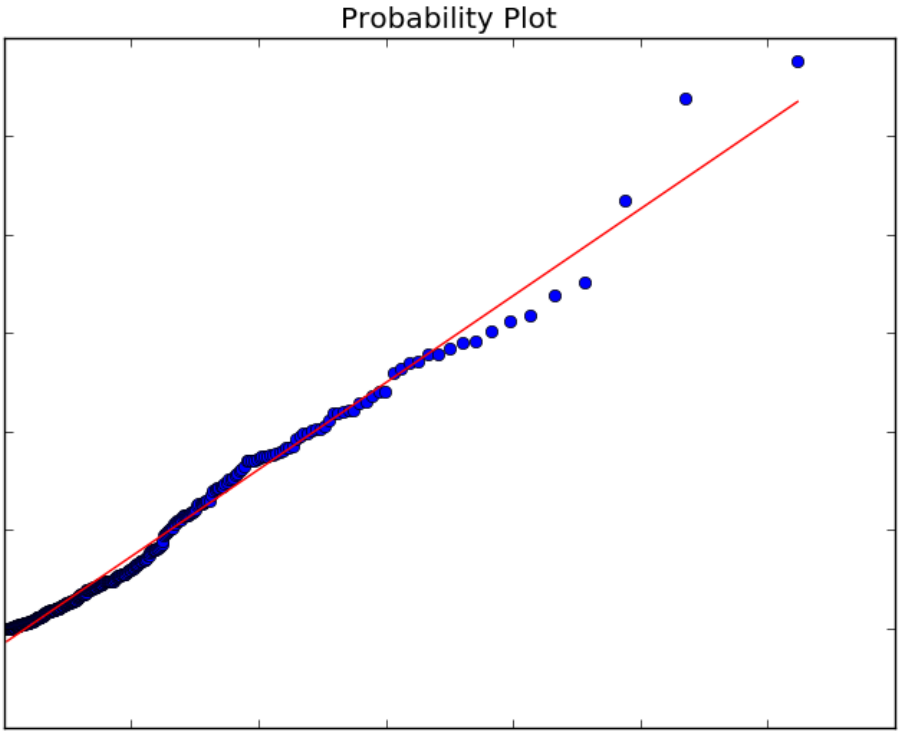

Figura 3.7: Q-Q Plot comparando um conteúdo muito popular com amostras de uma variável aleatória exponencial.

Por fim, repetimos o procedimento para conteúdos populares e pouco populares, chegando respectivamente aos valores de $1/\lambda$ igual a 350 e 10000. Os histogramas que foram usados para chegar a esses resultados se encontram na Figura 3.8 e na Figura 3.9.

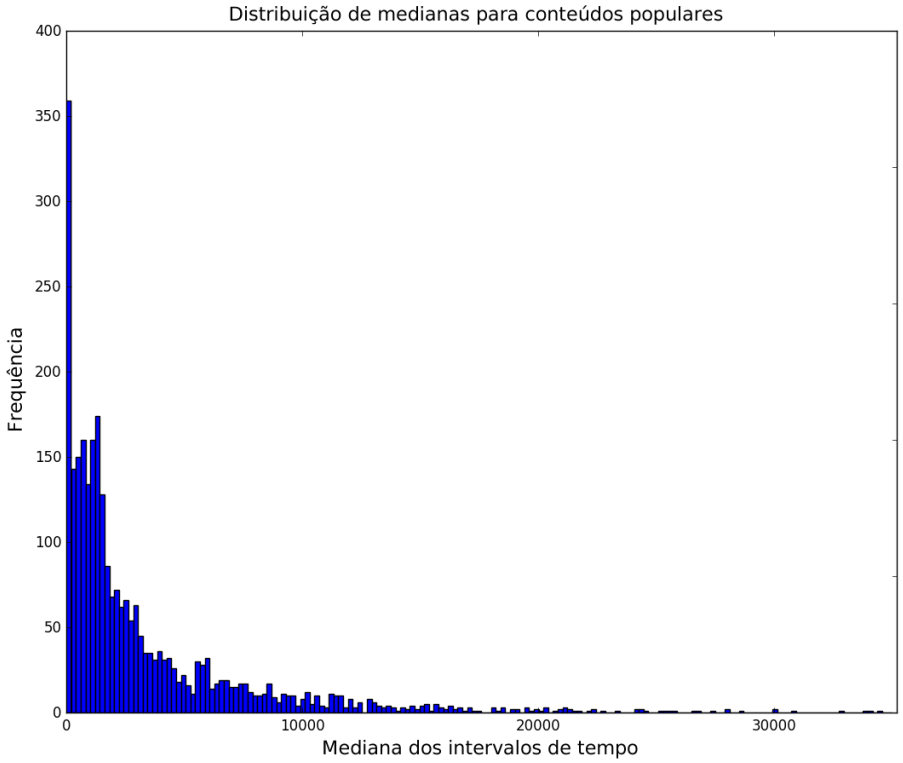

Figura 3.8: Histograma de medianas para conteúdos populares.

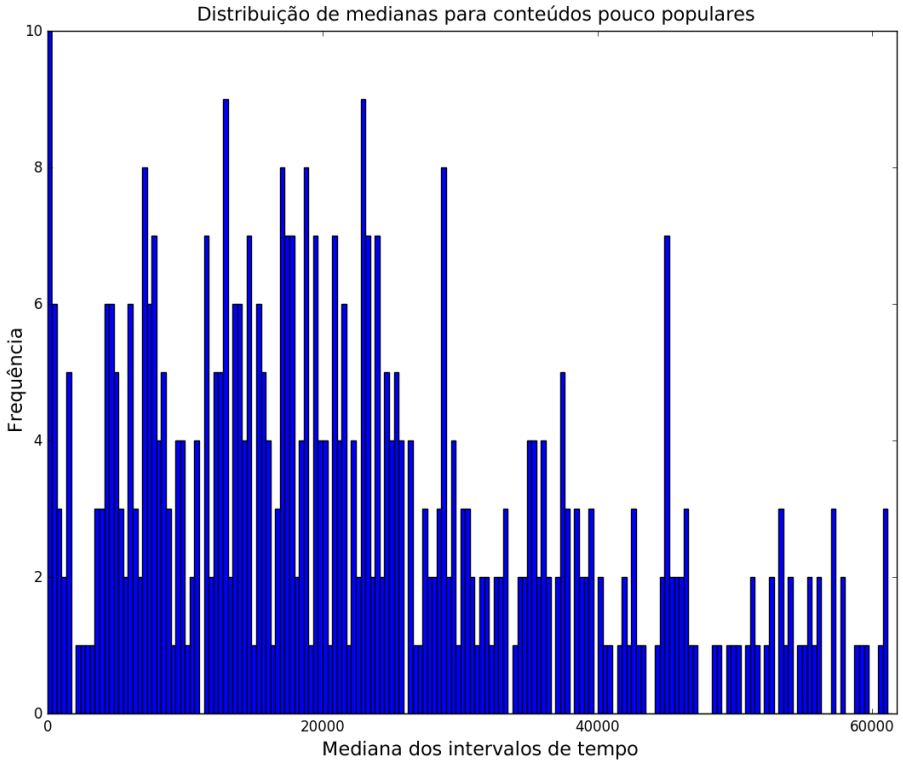

Figura 3.9: Histograma de medianas para conteúdos pouco populares.

Capítulo 4

Metodologia Usada para Simulação

Este capítulo abordará os passos seguidos na construção do simulador utilizado para realizar experimentos. Serão explicados os detalhes mais relevantes da implementação do simulador, assim como todos os parâmetros e métricas de interesse.

4.1 Estrutura

O simulador foi desenvolvido inteiramente na linguagem de programação C++, tendo como única dependência a biblioteca jsoncpp [10]. A biblioteca foi usada para facilitar a leitura do arquivo de parâmetros, assim como facilitar a escrita de resultados em disco, uma vez que o formato JSON é facilmente manipulável e possui estrutura e gramática fixas, ajudando tanto na compreensão dos arquivos quanto na organização deles.

A escala de tempo em que trabalharemos será a de minutos, por conta da escala de minutos que temos nos dados estudados no Capítulo 3. Sendo assim, partindo de um tempo zero de simulação, incrementaremos esse tempo usando números inteiros, de forma que um valor x indica que x minutos se passaram em tempo de simulação. Sendo assim, todos os parâmetros e eventos que se relacionam ao tempo de simulação devem estar expressos levando em conta a escala temporal em minutos.

O simulador foi dividido em quatro ”módulos”principais, cada um com uma função diferente na simulação. Temos o módulo central que é responsável pela leitura de parâmetros e validação dos mesmos, assim como pela organização da lista de eventos do simulador e pelo avanço da simulação conforme eventos são retirados da lista.

O módulo da topologia da rede controla o cache de cada nó da rede e armazena, para cada um deles, a informação referente a cada conteúdo, como o valor do RC e

a classe à qual aquele conteúdo pertence. Também é responsável por direcionar o passeio aleatório na rede, inserindo eventos na lista de eventos conforme necessário.

Os outros dois módulos são bem mais simples, um deles controla a geração de amostras de variáveis aleatórias, e o outro controla a geração de resultados após o término da simulação, processando as métricas de interesse de acordo com os estados dos nós da rede.

4.1.1 Lista de Eventos

Foi implementada uma estrutura de dados para representar a lista de eventos do simulador através de uma lista encadeada ordenada, pois não há uma estrutura definida na STL (Standard Template Library [11]) que funcione como uma lista encadeada ordenada. Sendo assim, optamos por implementar nossa própria estrutura ao invés de construir algo baseado no que se encontra na STL.

Uma solução fácil para evitar a implementação de uma estrutura de dados, seria usar uma lista fornecida pela STL e ordená-la a cada inserção, porém como o algoritmo de ordenação tem complexidade $O(n \log(n))$, se durante a execução do simulador tivermos m inserções nessa lista, a complexidade de manipulá-la seria da ordem de $O(mn \log(n))$, o que pode ser computacionalmente custoso se tivermos muitas inserções e/ou uma lista muito extensa.

Como utilizaremos essa lista para organizar a lista de eventos do simulador, precisamos que a inserção tenha a melhor complexidade possível, que no caso da lista encadeada ordenada, é de $O(n)$.

4.1.2 Cache

Para implementar o cache em cada nó da rede, foi criado um mapeamento entre o ID de cada conteúdo e um *struct* contendo as informações pertinentes àquele conteúdo. Um outro mapa é mantido com um ponteiro para as mesmas informações, porém somente daqueles conteúdos que residem no cache atualmente. Isso facilita o processamento dos conteúdos que estão de fato armazenados no cache. As informações associadas a cada conteúdos estão descritas abaixo:

- Identificador da classe de popularidade à qual o conteúdo pertence
- Valor do RC
- Contador de cache hits para este conteúdo neste cache
- *Timestamp* do último cache hit
- *Timestamp* da última inserção no cache

- *Timestamp* da última remoção do cache
- *Timestamp* em que o conteúdo será removido pela política TTL
- Vetor de amostras de tempo de vida no cache

As políticas de inserção e substituição no cache foram implementadas baseadas nos valores armazenados no *struct* descrito acima. O valor de RC está associado à inserção e remoção pela política RC, e a remoção pela política TTL está associada ao tempo de expiração guardado. Esse *timestamp* de expiração é gerado a partir de uma variável aleatória exponencial, com parâmetros discutidos abaixo na Subseção 4.3.1.

A política de troca por maior RC leva em conta os valores de RC armazenados para os conteúdos no cache. A política LRU leva em conta o *timestamp* do último cache hit. A política LFU leva em conta o valor do contador de cache hits para cada conteúdo do cache. A política FIFO leva em conta o *timestamp* da última inserção no cache para cada conteúdo que se encontra no cache.

4.2 Eventos

Nesta seção definiremos os eventos que coordenarão a simulação, assim como o papel de cada um deles e o impacto que têm na simulação. Definimos quatro eventos principais ao longo do processo de desenvolvimento do simulador, enumerados abaixo. Por simplicidade, referenciaremos os eventos pelos seus números no restante desta seção.

1. Geração de uma requisição por um cliente
2. Processamento de uma requisição por um nó da rede
3. Decréscimo do valor de RC
4. Expiração do conteúdo armazenado em cache pela política TTL

A geração do evento de tipo 1 é feita a partir do banco de dados analisado no Capítulo 3. Observamos que cada entrada do banco corresponde a uma requisição por um conteúdo, que será então traduzida para um evento de requisição no simulador. Por comodidade, todos os eventos de requisição serão inseridos na fila de eventos antes do início da simulação, portanto iniciaremos nossa simulação com mais de 1 milhão de eventos já na fila, e terminaremos quando todos os eventos deste tipo forem atendidos.

O evento de tipo 1 está associado aos seguintes dados: um *timestamp* em tempo de simulação, um ID de conteúdo, o ID da classe de popularidade à qual o conteúdo

pertence, e o ID do cliente associado à requisição original. Como temos uma grande quantidade de clientes cadastrados no banco, e trabalharemos com uma topologia simplificada com apenas alguns nós, precisaremos associar cada cliente do banco a um cliente da topologia. Por conta da diferença numérica, vários clientes do banco estarão associados ao mesmo cliente da topologia, e essa associação será feita de forma aleatória.

O evento de tipo 1 deverá gerar um evento de tipo 2, associado ao nó da topologia em que o cliente está conectado. O tempo deste próximo evento será o mesmo do evento original, pois descartaremos o atraso da comunicação entre nós da rede, conforme explicado na Seção 4.1.

O evento de tipo 2 é tratado pelos nós intermediários da rede, que seguirão um procedimento para alterar seu estado de cache baseado na requisição que está sendo atendida. Um fluxograma explicando o funcionamento básico deste procedimento pode ser visto na Figura 4.1.

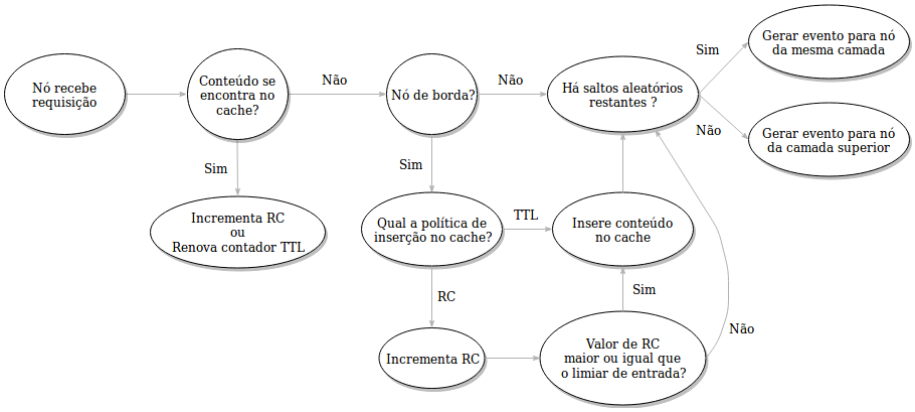

Figura 4.1: Fluxograma descrevendo o procedimento executado no evento de processamento de requisição.

Comentando sobre alguns detalhes no fluxograma, se o conteúdo se encontra no cache, simplesmente incrementamos o RC ou renovamos o tempo de expiração da política TTL, de acordo com a política empregada na simulação. Não precisamos continuar transmitindo essa requisição pela rede pois conseguimos servir o conteúdo para o cliente a partir do momento que encontramos o conteúdo em algum cache da rede.

A definição de ”nó de borda”do fluxograma foi explicada na Subseção 2.1.2, trata-se de um nó que recebeu uma requisição que saiu de uma camada inferior à que se encontra. O evento do tipo 2 guarda por quantos nós da camada atual a requisição já passou, com o objetivo de decidir sobre a continuação ou não do passeio aleatório, encaminhando a requisição para algum nó da camada apropriada. Se o destino for da mesma camada, é sorteado algum de seus vizinhos de maneira uniforme, e é gerado então um outro evento do tipo 2 para o nó da mesma camada

sorteado. Porém, se o destino for a camada superior, é gerado um evento do tipo 2 para o único nó dessa camada ao qual o nó atual está conectado.

Uma última observação sobre o fluxograma, o passo de "inserir o conteúdo no cache"deve remover algum conteúdo do mesmo caso não haja mais espaço. Para tanto, será utilizada a política de remoção vigente durante a simulação.

O evento de tipo 3 é gerado quando algum conteúdo é inserido em qualquer cache. Ao ser processado, ele decrementa o valor de RC de todos os conteúdos da mesma classe do conteúdo que gerou esse evento, e remove aqueles cujo valor de RC atingiu o limiar de saída do cache. Se ainda há conteúdos daquela classe no cache, outro evento do tipo 3 é agendado conforme os parâmetros de decréscimo do RC. Se ao inserir um conteúdo no cache, já houver um evento de decréscimo de RC já agendado para a classe daquele conteúdo para este cache, não é agendado um novo evento. Sendo assim, podemos ter no máximo $N * C$ eventos de decréscimo de RC na fila ao mesmo tempo, onde N representa o número de nós na rede e C o número de classes de popularidade considerados.

O evento do tipo 4 se assemelha ao do tipo 3, ele foi definido para o caso da política de inserção no cache ser a TTL ao invés da RC. O evento deve retirar do cache o conteúdo que deu origem ao evento, pois seu tempo no cache foi expirado. Se, porém o conteúdo no cache receber alguma requisição enquanto este evento ainda está na fila, ele deve ser descartado da mesma e reagendado, efetivamente renovando o tempo de expiração. Como reordenar a fila de eventos ao alterar o tempo de um evento é uma operação custosa, esse evento não foi implementado dessa maneira no simulador, ainda mais pelo fato de que teríamos $N * K$ eventos, onde N representa o número de nós na rede e K o número de conteúdos considerados.

Ao invés disso, a abordagem tomada foi de armazenar junto ao cache o tempo em que aquele conteúdo terá expirado, e ao receber uma requisição por um conteúdo, é verificado antes de qualquer outra coisa se algum conteúdo no cache expirou, e em caso positivo, remove-lo. Dessa forma, evitamos poluir a lista de eventos com um número considerável de eventos de expiração, e reduzimos assim a complexidade computacional do simulador.

4.3 Parâmetros

O simulador conta com um conjunto de parâmetros que controlam os eventos descritos acima e descrevem as condições de rede que estamos simulando. Nesta seção focaremos nos três grupos de parâmetros que regem o funcionamento do simulador: os parâmetros gerais (ou de controle de eventos), a topologia da rede utilizada e o traço de simulação utilizado para gerar eventos.

4.3.1 Parâmetros Gerais

Os parâmetros ditos ”gerais”se referem a parâmetros que controlam eventos locais no simulador e modificam o funcionamento das rotinas que tratam esses eventos. Abaixo encontramos uma listagem desses parâmetros, junto do seu significado e seu papel na simulação:

- **Tempo limite de simulação**: Controla o tempo de simulação a partir do qual eventos que ainda estão na lista de eventos serão ignorados e descartados, para dar fim à simulação. Como temos 90 dias de dados, conforme descrito no Capítulo 3, tipicamente este parâmetro estará fixo em 129600, a quantidade de minutos contidos num período de 90 dias.

- **Número de conteúdos**: Controla o limite de IDs a serem distribuídos para os conteúdos requisitados, e será igual a 5267, o número de conteúdos encontrados na análise feita no Capítulo 3.

- **Número de classes de conteúdos**: Controla em quantas classes os conteúdos estão divididos, para corretamente manusear eventos e métricas de interesse. Mais uma vez, conforme analisado no Capítulo 3, este será fixo em 3.

- **Número de saltos no passeio aleatório**: Este parâmetro controla o número máximo de saltos dados em cada passeio aleatório em cada camada da topologia utilizada. Uma vez que o número de saltos dados se iguale ao descrito por este parâmetro, a requisição será enviada à camada superior.

- **Tamanho do cache**: Este parâmetro controla o espaço disponível em cada cache da rede. Assumimos que cada conteúdo terá o mesmo tamanho, portanto podemos definir este parâmetro como o número de conteúdos que podem ocupar o cache concorrentemente.

- **Política de inserção no cache**: Alguma das políticas de inserção apresentadas na Subseção 2.2.1. Controlará como os conteúdos serão inseridos no cache.

- **Política de remoção do cache**: Alguma das políticas de remoção apresentadas na Subseção 2.2.2. Controlará como os conteúdos serão substituídos no cache quando este estiver cheio.

- **Limiar de entrada do RC**: Quando a política de inserção é a política RC, este parâmetro controla o valor que o RC deve atingir para que o conteúdo seja inserido no cache. Uma vez que atinja este valor, o conteúdo é inserido no cache.

- **Limiar de saída do RC**: Quando a política de inserção é a política RC, este parâmetro controla o valor que o RC deve atingir para que o conteúdo seja removido no cache. Uma vez que o RC atinja este valor, o conteúdo é removido do cache.

- **Taxa de decréscimo do RC para cada classe de conteúdo (μ)**: Este conjunto de parâmetros informa ao simulador a taxa a ser utilizada ao amostrar a variável aleatória exponencial que rege o decréscimo do valor do RC e a expiração de conteúdos na política TTL. É calculado com base nos valores de $1/\lambda$ calculados na Seção 3.5.

O valor de μ é calculado com base no valor de $1/\lambda$ de cada classe de conteúdos, conforme mencionado na Seção 3.5, além do valor k que é o limiar de entrada do RC. A fórmula para o cálculo é a seguinte [6]:

$$\pi_C = \left(\frac{\lambda}{\mu}\right)^k ; \mu = \frac{\lambda}{\sqrt[k]{\pi_C}}$$

Onde π_C é a probabilidade desejada de algum conteúdo da classe C estar presente no cache em algum momento, que foi fixo nos seguintes valores para este projeto:

- **Muito populares**: $\pi_C = 0.9$

- **Populares**: $\pi_C = 0.7$

- **Pouco populares**: $\pi_C = 0.5$

Observamos que para parametrizar a política TTL, assumimos que $k = 1$, o menor valor possível, pois já com uma requisição o conteúdo deve ser inserido no cache, conforme previsto na política TTL.

4.3.2 Traço de Simulação

O traço de simulação é um dos parâmetros do simulador que controlará como os primeiros eventos serão gerados. No caso, ele é responsável por registrar todos os eventos do tipo 1 descritos na Seção 4.2, assim como todas as informações pertinentes a ele, como ID do conteúdo, ID da classe de conteúdos e ID do cliente que deu origem à requisição, assim como sua região.

Esse arquivo contendo essas informações deve portanto ser baseado nos dados estudados no Capítulo 3, em que cada linha deve corresponder a uma entrada do banco, com seus dados transformados de acordo com o método de estudo proposto.

4.3.3 Topologia

A topologia usada deve descrever todos os nós pertencentes à rede estudada, incluindo com quem está conectado e quem está conectado a ele. Assume-se que essa topologia é hierarquizada, ou seja, os nós estão divididos em camadas, e nós podem estar conectados em outros da mesma camada ou de uma camada superior, de maneira que uma requisição possa trafegar da camada mais baixa para a camada mais alta. Assume-se também que todos os nós da camada mais superior estão conectados a uma área de publicação de conteúdos, em que todos os conteúdos estão armazenados em disco e podem ser servidos para os clientes.

Para os experimentos descritos neste trabalho, nos limitamos a uma topologia idealizada em que os nós estariam distribuídos de maneira balanceada entre as camadas, de forma que a hierarquia se assemelharia a uma árvore cuja raiz seria a área de publicação. Para nos aproveitarmos da separação em regiões observada nos dados descritos no Capítulo 3, separaremos a topologia em três regiões desconexas entre si, de maneira a simular uma desconexão geográfica entre as regiões.

Além disso, assumiremos que os nós que se encontram na mesma região e na mesma camada formarão um (ou mais) grafo(s) completo(s), de maneira que o passeio aleatório possa visitar cada um dos nós no mesmo ”hub”de cada camada antes de se dirigir à camada superior. Na Figura 4.2 vemos uma ilustração da rede formada por um grafo completo de 5 nós.

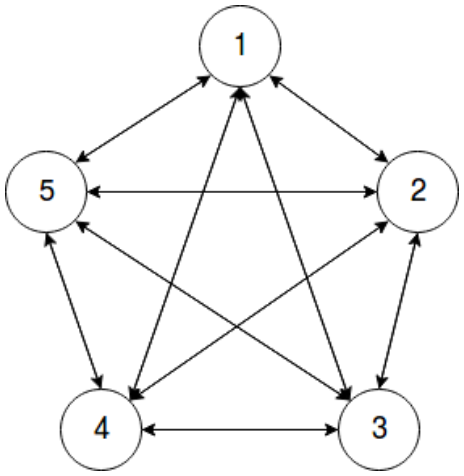

Figura 4.2: Esquema de um grafo completo com 5 vértices

Assumindo a notação de que um grafo completo com N vértices pode ser escrito como K_N, podemos detalhar a rede utilizada nos experimentos deste trabalho conforme demonstrado na Figura 4.3. Os valores numéricos se baseiam na divisão do número de clientes por região descrita na Seção 3.2.

Lembramos que o mapeamento entre os clientes do banco de dados e os da topologia acima não pode ser uma simples associação 1:1 em que cada cliente do banco

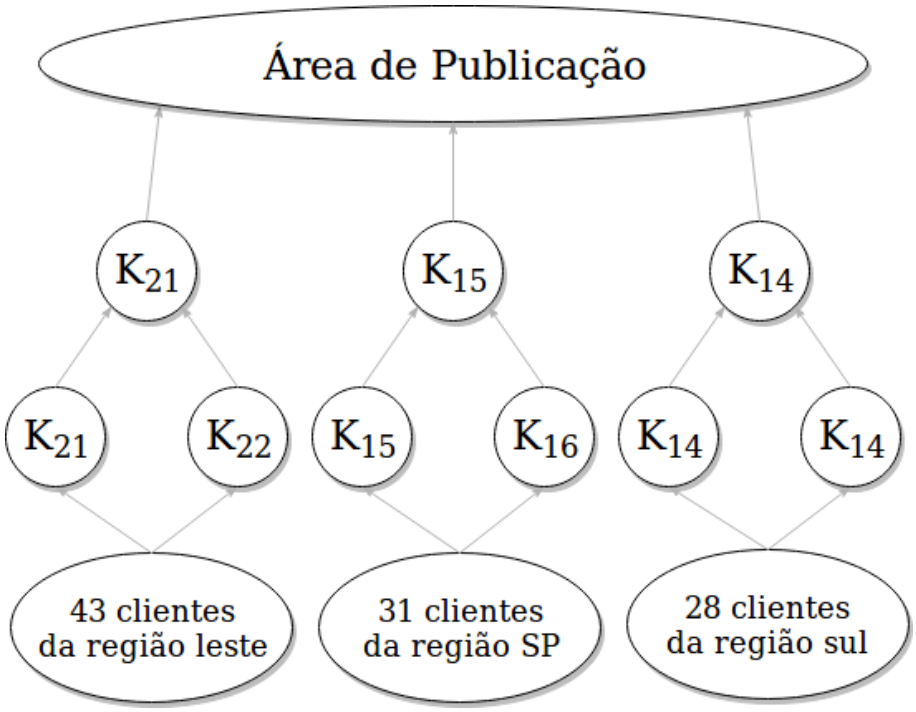

Figura 4.3: Esquema da topologia utilizada.

corresponde a um cliente da topologia, conforme explicada na Seção 4.2. O afunilamento da topologia à medida que nos aproximamos da área de publicação busca simular o fato de que em uma rede hierárquica, as camadas mais altas costumam possuir menos nós que as mais baixas.

4.4 Métricas de Interesse

Para analisar o desempenho da rede, adotamos quatro métricas principais. Descrevemos abaixo o propósito de cada uma e como calcular cada uma delas a partir dos eventos tratados pelo simulador:

Fração de requisições filtradas pela rede: Cada evento de requisição por um cliente pode ser atendida por qualquer nó intermediário da rede, dado que o conteúdo requisitado esteja armazenado no seu cache. Se, ao longo dos passeios aleatórios, o conteúdo não for encontrado em nenhum cache, a requisição será atendida na área de publicação, caracterizando-a como uma requisição não filtrada. Outras maneiras de chamar essa métrica incluem "Carga filtrada pela rede"e "Probabilidade de Hit em algum cache da rede". Assim, definimos a fração F de requisições filtradas a partir do total de requisições R e do total de requisições não filtradas R_{server} como:

$$F = 1 - \frac{R_{server}}{R}$$

Número médio de escritas no cache: Toda vez que um conteúdo é adicionado

ao cache, independente de estar substituindo outro conteúdo ou não, realizamos uma escrita no cache. A fim de contabilizar quantas escritas cada cache realizará, em média, ao longo da simulação, cada cache mantém um contador de escritas que é incrementado toda vez que um conteúdo é adicionado ao cache. Ao final da simulação, tiramos a média deste contador dentre todos os caches da rede. Portanto, sendo N o número de caches na rede e W o número de escritas feitas em um cache ao longo da simulação, o número médio de escritas W_m será:

$$W_m = \sum_{\forall i \in N} W_i / |N|$$

Ocupação do cache: Ao final da simulação, contamos quantos conteúdos de cada classe de popularidade encontram-se em cada cache, e tiramos a média dentre todos os caches. Com isso, obtemos a ocupação média do cache por classe de popularidade, e com ela a ocupação média geral do cache. É importante ressaltar que essa métrica referencia somente o estado da rede no final da simulação, e portanto não representa bem o comportamento da rede em qualquer instante da simulação. Assim, se representarmos por $U_{i,j}$ o número de conteúdos presentes no cache i pertencentes à classe j, a ocupação média U_m total será:

$$U_m = \sum_{\forall i \in N} \left(\sum_{\forall j \in C} U_{i,j} / |N| \right) / |C|$$

Tempo médio de vida no cache: Uma vez que um conteúdo seja armazenado no cache, ao ser removido o seu tempo de vida no cache é contabilizado. Ao final da simulação, cada conteúdo em cada nó da rede terá um conjunto T de valores para seu tempo de vida no cache. Para visualizar essa métrica mais facilmente, tiramos a média dessas amostras para cada conteúdo K que pertence a uma mesma classe C de popularidade. Para termos então uma métrica global V_m da rede, tiramos a média para todo nó N da rede. Definindo então por $t_{i,j,k}$ a i-ésima amostra de tempo do conteúdo j no nó k da rede, podemos escrever a métrica como:

$$V_m = \sum_{\forall k \in N} \left(\frac{\sum_{\forall j \in C} \sum_{\forall i \in T} t_{i,j,k}}{|C| * \sum_{\forall j \in C} |T_j|} \right) / |N|$$

Capítulo 5

Experimentos e Resultados

Este capítulo apresenta alguns experimentos realizados com o simulador descrito no capítulo anterior. A cada experimento apresentado será descrito qual o conjunto de parâmetros utilizado, e como cada métrica de interesse se comportou ao variarmos cada parâmetro.

Cada seção deste capítulo relata os resultados relevantes à variação do parâmetro que dá nome à seção, mantendo todos os outros parâmetros fixos, a fim de isolar variáveis para melhor interpretar os resultados obtidos. Serão apresentados gráficos com o comportamento das métricas de interesse, acompanhados da sua análise com base na teoria descrita nos capítulos anteriores.

5.1 Descrição dos Experimentos

5.1.1 Parâmetros Usados na Simulação

Abaixo destacamos o conjunto de parâmetros padrão para o simulador. Qualquer parâmetro que não for explicitamente alterado na entrada assumirá o valor abaixo. O significado de cada parâmetro foi descrito no capítulo anterior.

- Número de saltos no passeio aleatório: **2**
- Tamanho do cache de cada nó da rede: **20**
- Valor do RC de entrada: **1**
- Valor do RC de saída: **0**
- Política de inserção no cache: **RC**
- Política de remoção no cache: **LRU**
- $1/\mu$ para conteúdos muito populares: Calculado pela fórmula na Subseção 4.3.1, igual a **30** para $k = 1$

- $1/\mu$ para conteúdos populares: Calculado pela fórmula na Subseção 4.3.1, igual a **271** para $k = 1$
- $1/\mu$ para conteúdos pouco populares: Calculado pela fórmula na Subseção 4.3.1, igual a **7071** para $k = 1$
- Topologia da rede: Descrita na Subseção 4.3.3

5.1.2 Apresentação dos Resultados

Cada simulação executada gera um arquivo de resultados contendo os valores finais de cada métrica observada. Para cada configuração de parâmetros, a simulação é repetida 20 vezes, e é feita uma média dos valores obtidos para chegar a um valor final para cada métrica, acompanhado do desvio padrão observado entre as 20 amostras. Os resultados apresentados nas seções a seguir tomam como base esses valores médios acompanhados dos seus desvios padrões, que podem ser vistos graficamente, exceto quando a escala dos gráficos apresentados tornar o valor do desvio padrão pequeno o suficiente para ser imperceptível.

5.2 Tamanho do Cache

Mantendo os valores padrão para os parâmetros de simulação, variamos o tamanho do cache para analisar o efeito deste parâmetro nas métricas de interesse. Foram tomadas três faixas de valores para serem observados, descritas abaixo:

- **Caches pequenos**: Valores de 1 a 20, incrementados 1 a 1
- **Caches médios**: Valores de 20 a 50, incrementados 5 a 5
- **Caches grandes**: Valores de 50 a 200, incrementados 50 a 50

Observando a Figura 5.1, notamos que a fração de requisições filtradas cresce rapidamente com o tamanho do cache até certo valor (por volta de 50), e então permanece quase constante para tamanhos de cache maiores. Para conteúdos pouco populares, precisamos de um cache maior para atingir o valor constante observado nas demais curvas, pelo fato de precisarmos de um cache maior para acomoda-los sem criar disputa por espaço do cache, que inevitavelmente trocaria um conteúdo pouco popular por algum outro mais popular.

Podemos explicar o crescimento rápido observado pelo fato de que com mais espaço no cache, teremos menos disputa por esse espaço pelos conteúdos, que poderão então usufruir de mais tempo no cache, suficiente para atender novas requisições e ajudar na filtragem de requisições na rede. A Figura 5.2, que mostra o

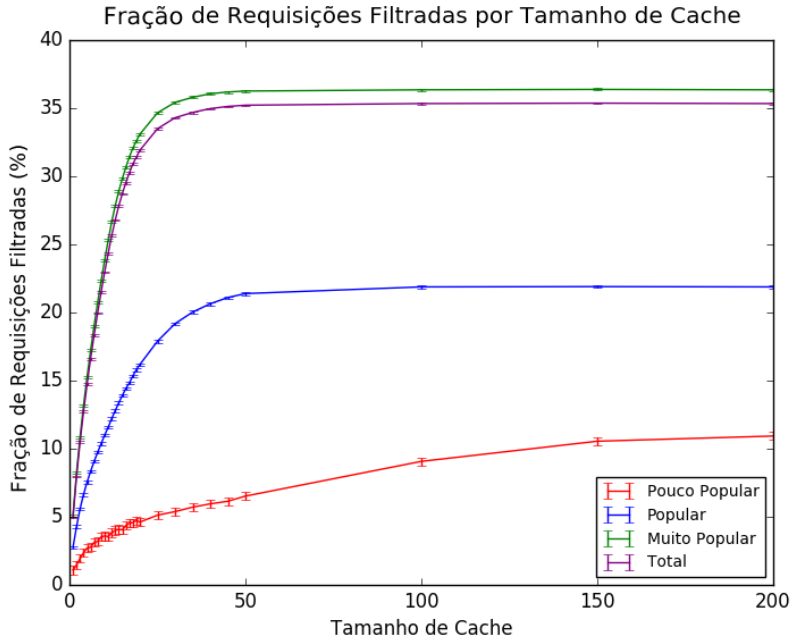

Figura 5.1: Fração de requisições filtradas, por tamanho de cache.

tempo médio de vida no cache para conteúdos populares e muito populares, segue o mesmo comportamento observado na fração de requisições filtradas.

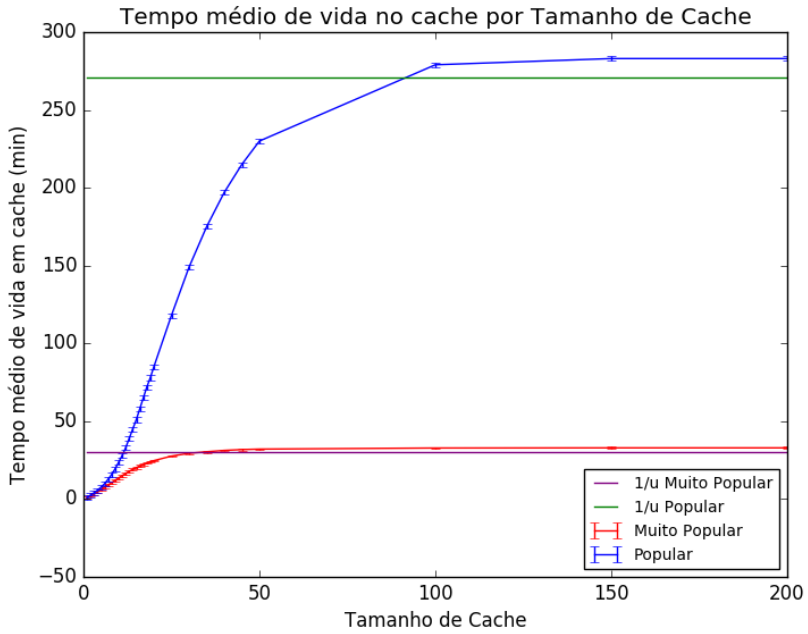

Figura 5.2: Tempo médio de vida no cache, por tamanho de cache.

Outra observação importante na Figura 5.1 é que o percentual total de requisições filtradas é menor do que o referente a conteúdos muito populares. Isso acontece pois o valor dito *total* é uma soma condicionada a cada popularidade, ou seja, como

as requisições por conteúdos populares e pouco populares foram menos filtradas, o percentual total cai junto dessas duas métricas.

A estabilização da fração de requisições filtradas observada na Figura 5.1 pode ser explicada pelo fato de conteúdos não permanecerem sempre no cache, mesmo em condições de concorrência quase nula. O valor do contador de requisições (RC) é decrementado em intervalos de tempo aleatórios, e se não for incrementado por novas requisições provocará a saída do conteúdo do cache. Dessa forma, por mais que tenhamos espaço em cache de sobra, ele não será completamente utilizado devido ao mecanismo RC. A Figura 5.3 mostra o quanto do cache está sendo ocupado por cada classe de popularidade ao término da simulação. Nela podemos observar que à medida que aumentamos o tamanho do cache, menor a porcentagem dele que conseguimos ocupar, indicando que a partir de certo tamanho de cache deixamos de melhorar o desempenho da rede, justamente devido ao mecanimso RC.

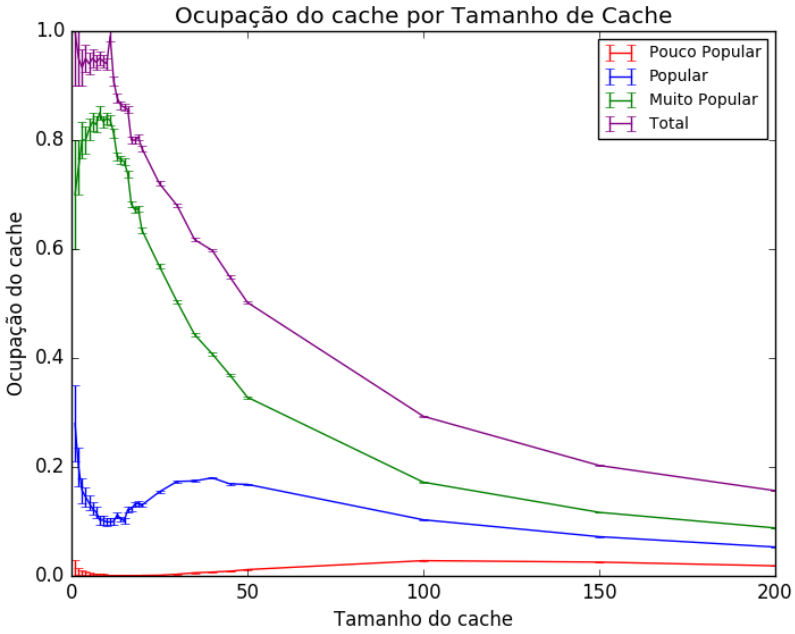

Figura 5.3: Ocupação do cache, por tamanho de cache.

Ainda na Figura 5.3, observamos que conteúdos pouco populares só passam a ter algum espaço no cache se o tamanho do cache for grande o suficiente para remover boa parte da competição pelo seu espaço, dando oportunidade de conteúdos menos populares se instalarem nele por algum tempo. Com caches muito pequenos, predominam os conteúdos muito populares, e à medida que aumentamos o espaço do cache, conteúdos de popularidade média podem se instalar e manter-se no cache, por conta da competição reduzida pelo espaço, como podemos ver pelo início das curvas azul e verde.

A última métrica que iremos abordar nesta seção é o número de escritas no

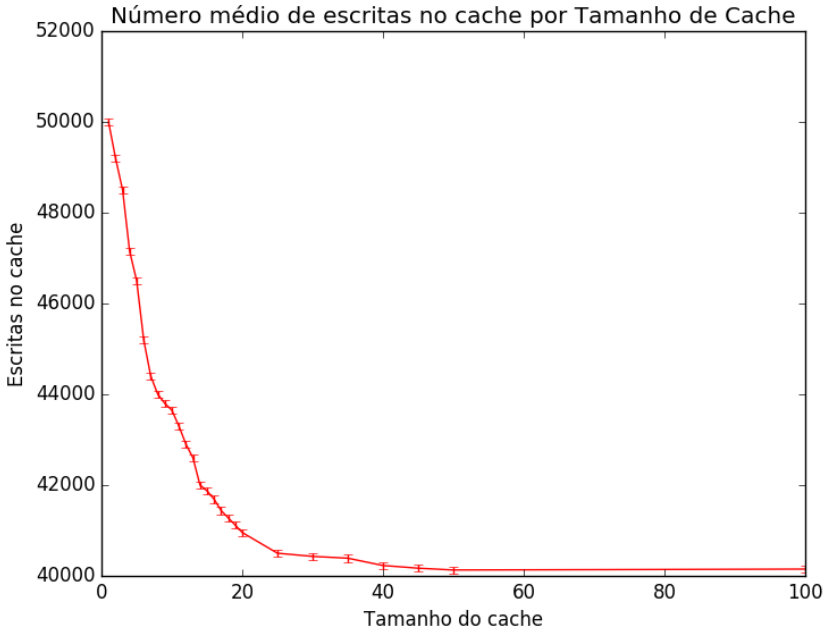

Figura 5.4: Número de escritas no cache, por tamanho de cache.

cache, que podemos analisar pela Figura 5.4. Conforme aumentamos o tamanho do cache, diminuímos a disputa por espaço de cache pelos conteúdos, o que por sua vez diminui o número de trocas que devem ser realizadas no cache, diminuindo então o número de escritas. Esse valor estabiliza da mesma forma que a fração de requisições filtradas na rede, conforme vimos na Figura 5.1, justamente pois a partir de certo tamanho de cache, a disputa por espaço é quase nula.

Um ponto importante a ser ressaltado na Figura 5.2 é o fato de que o tempo médio de vida no cache para conteúdos muito populares é, em média, muito inferior ao de conteúdos populares, porém os conteúdos muito populares têm uma fração de requisições filtradas muito maior. Para justificar essa diferença, voltamos aos parâmetros descritos na Subseção 5.1.1. Em particular, aos parâmetros $1/\mu$ de cada classe de popularidade, que indicam a média da variável aleatória exponencial que controla o decréscimo do RC.

Observamos que as curvas tendem a se estabilizar nos valores equivalentes aos parâmetros $1/\mu$, o que indica que, em média, os conteúdos saem após 1 decréscimo do RC, que seria um tempo baixo demais para justificar uma fração de requisições filtradas tão alta. Para entender esse comportamento, calculamos a distribuição do tempo de vida em cache ilustrado na Figura 5.5. Nela podemos ver que uma quantidade considerável de amostras são maiores do que a média observada, e que chegamos a ter valores 20 vezes maiores que a média da variável aleatória.

Para entender o valor da média dessa métrica, devemos atentar ao fato de que se um conteúdo muito popular recebe N requisições espaçadas, há uma grande

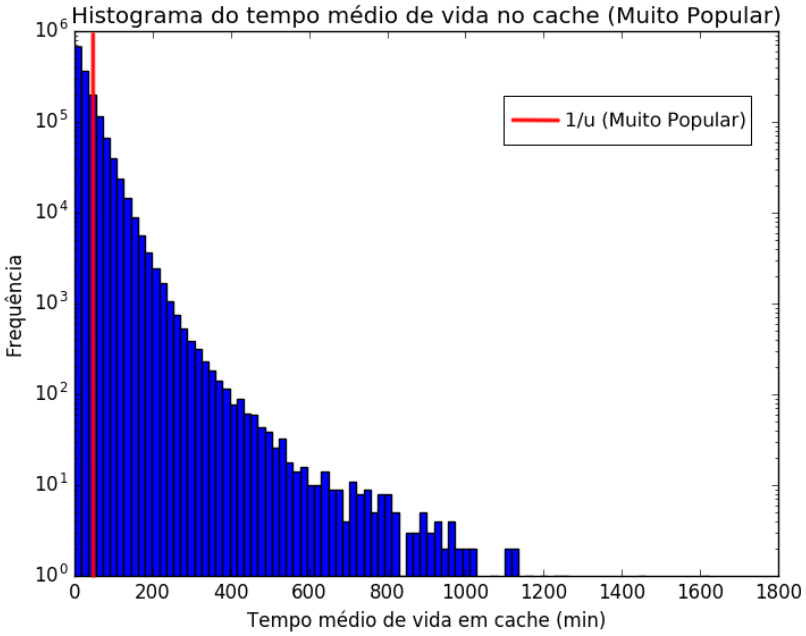

Figura 5.5: Histograma do tempo de vida em cache, em escala log.

chance de ele entrar e sair do cache N vezes, gerando portanto N amostras de valor (em média) $1/\mu$ para a métrica tempo de vida no cache. Porém, se essas N requisições chegarem pouco espaçadas, em uma rajada, há uma grande chance do conteúdo entrar e sair do cache apenas 1 vez, gerando 1 amostra de valor (em média) N/μ. Isso portanto favorece o fato da média se aproximar tanto do valor de $1/\mu$. Dessa forma, por mais que o tempo médio de vida no cache seja baixo, há diversos conteúdos muito populares que ficam mais tempo no cache, e ajudam a fração de requisições filtradas a se manter elevada.

De maneira similar, podemos observar a distribuição do tempo de vida no cache para conteúdos populares, cujo valor de $1/\mu$ é diferente do de muitos populares, na Figura 5.6. Observamos que há amostras de valores 13 vezes maiores que o valor da média, porém essas amostras têm pouco impacto na média.

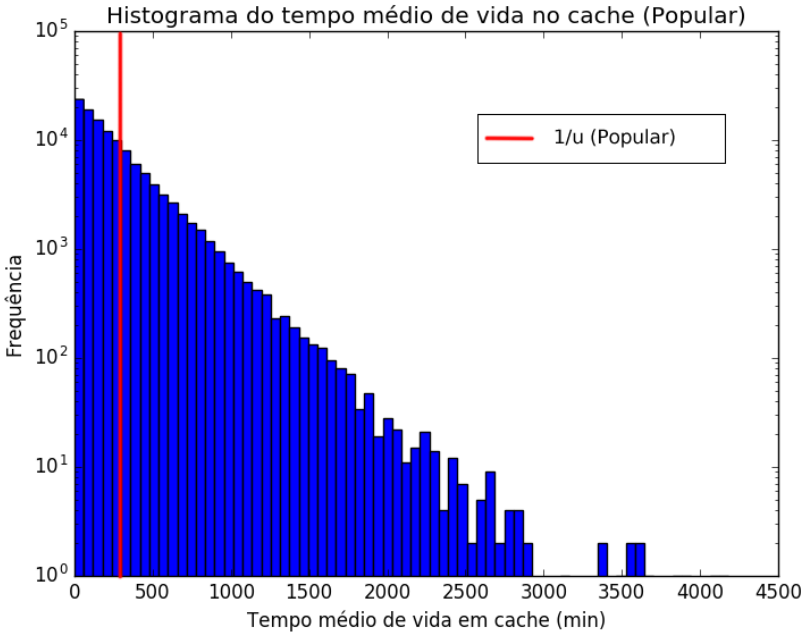

Figura 5.6: Histograma do tempo de vida em cache, em escala log.

5.3 Número de Saltos Aleatórios

Mantendo os valores padrão para os parâmetros de simulação, variamos nesta seção o número de saltos aleatórios feitos durante o passeio aleatório, para analisar o efeito deste parâmetro nas métricas de interesse. Este valor foi variado de 1 até 10, com intervalos de tamanho 1, e os resultados se encontram na análise a seguir.

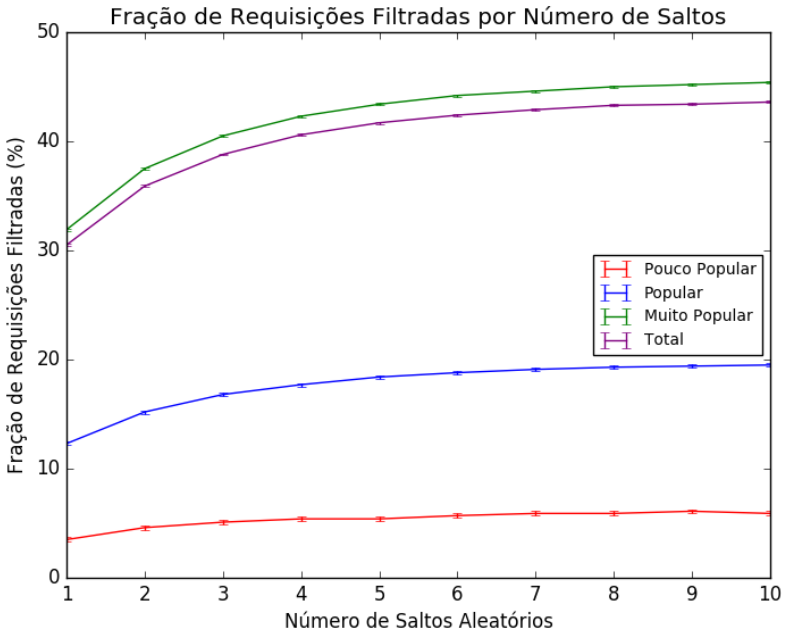

Figura 5.7: Fração de requisições filtradas, por número de saltos.

Podemos observar na Figura 5.7 que à medida que aumentamos o número de saltos do passeio aleatório, aumentamos a fração de requisições filtradas, para todas as faixas de popularidade. Para explicar esse aumento, notamos que ao passarmos mais tempo buscando o conteúdo em uma camada da rede, aumentamos a chance de chegar aleatoriamente a algum cache que tenha esse conteúdo. Esse crescimento, porém, não é linear, e tende a diminuir conforme aumentamos muito o número de saltos, pois se o conteúdo não encontra-se em nenhum cache da camada, buscá-lo em vários caches não trará benefícios.

Se considerarmos o passeio aleatório com o número de saltos tendendo a infinito, teremos o mesmo valor do número de requisições filtradas caso realizemos uma busca determinística a todos os caches da camada observada.

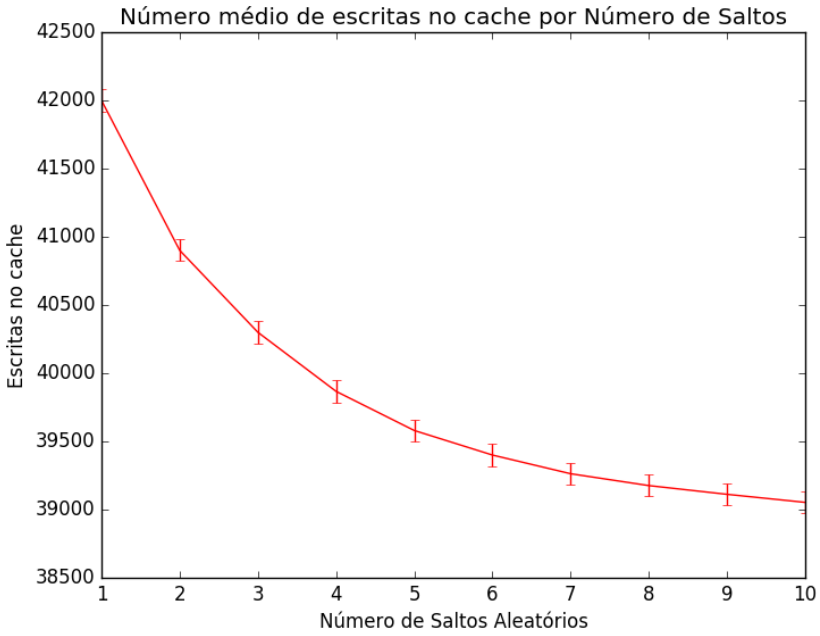

Figura 5.8: Número de escritas no cache, por número de saltos.

Com o aumento do número de saltos, existe uma chance maior de encontrar o conteúdo requisitado em algum cache da camada, e portanto incrementar seu RC. O conteúdo ficará um tempo maior armazenado no cache, evitando possíveis reinserções. Como consequência temos o crescimento da fração de requisições filtradas observado na Figura 5.7 e um decréscimo no número de escritas no cache ilustrado na Figura 5.8.

Observando, por fim, o tempo médio de vida no cache, vemos que ele se mantém praticamente constante para conteúdos populares e muito populares, com o aumento do número de saltos. No caso dos muito populares, observamos que o tempo médio se aproxima do valor do parâmetro $1/\mu$, pelos mesmos motivos observados e explicados na Figura 5.5. Já para para conteúdos populares o valor é bem abaixo do parâmetro

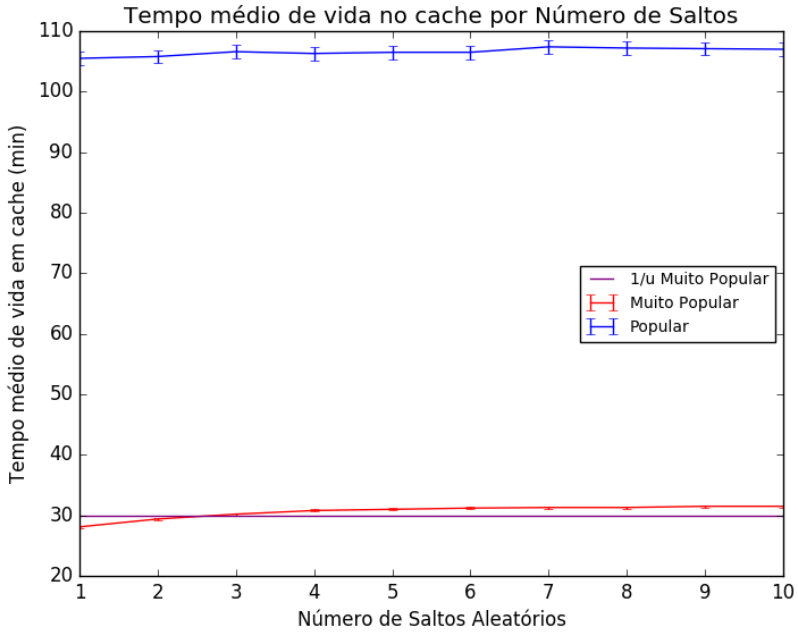

Figura 5.9: Tempo médio de vida no cache para conteúdos muito populares, por número de saltos.

$1/\mu$ dessa classe. Conforme observado na Figura 5.2, o tempo médio de vida no cache se aproxima do parâmetro $1/\mu$ para valores de cache superiores a 100 e neste experimento consideramos o cache igual 20.

Pelos resultados obtidos para a métrica tempo médio de vida no cache, poderíamos concluir que aumentar o número de saltos aleatórios não melhora o desempenho do sistema. A princípio essa conclusão parece contradizer o resultado das Figura 5.7 e Figura 5.8. Porém lembramos que esta métrica é uma média dos tempos de vida no cache, e por mais que alguns conteúdos consigam ficar muito tempo no cache, ao calcularmos a média eles terão seu peso bem reduzido, conforme explicado na Figura 5.5.

Ao aumentarmos o número de saltos aleatórios, estamos beneficiando conteúdos que não conseguiam ficar no cache por muito tempo pois o passeio aleatório limitado não permitia que eles fossem encontrados. O fato desses conteúdos permanecerem no cache por mais tempo, melhora o desempenho das métricas fração de requisições filtradas e número de escritas no cache. Conteúdos que sairiam do cache rapidamente (taxa de chegada de requisições muito baixa) ou ficariam no cache por muito tempo (taxa de chegada de requisições muito alta) terão suas métricas praticamente inalteradas com o aumento do número de saltos.

5.4 Valor do RC de Entrada e Saída

Nesta seção, variaremos os valores de entrada e saída do RC, mantendo os demais parâmetros inalterados, para verificar como o valor definido pelo RC afeta as métricas de interesse. Notamos que os valores de entrada e saída variarão mantendo uma diferença de 1 entre eles, pois não queremos avaliar nesta seção como a histerese afeta as métricas. Variaremos então os valores de entrada de 1 a 10, e os de saída de 0 a 9, ambos com incremento de 1.

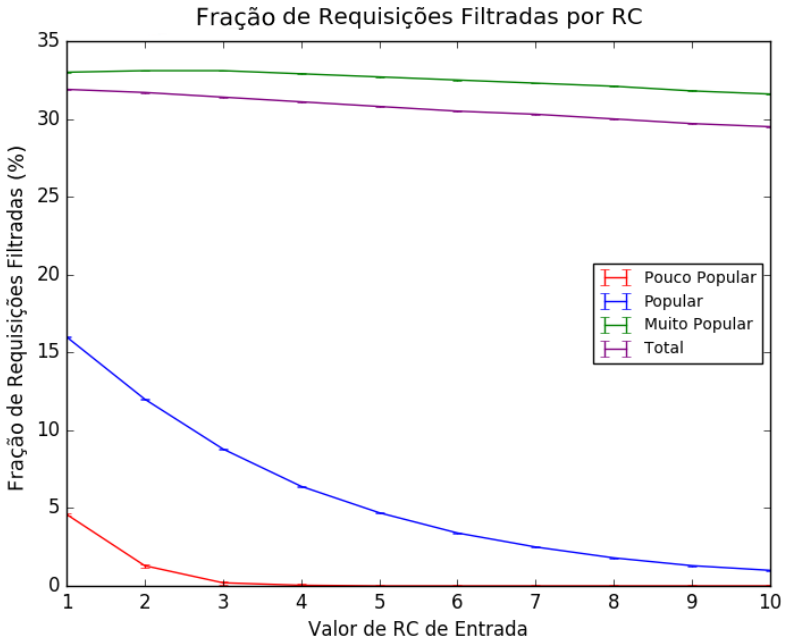

Figura 5.10: Fração de requisições filtradas, por valor de RC.

Observamos na Figura 5.10 que conforme aumentamos o valor do RC para entrar (e sair) do cache, diminuímos a fração de requisições filtradas pela rede, independente de popularidade. Podemos explicar esse resultado pois se o RC possui um limiar maior para entrada no cache, menos conteúdos conseguirão entrar de fato no cache. Notamos que os conteúdos de popularidade média e baixa são os mais afetados, pois as taxas médias de chegada de requisições e de decréscimo do RC não permitem que o limiar do RC para entrada no cache seja alcançado. Além disso, é importante lembrar que a curva que representa a fração total de requisições filtradas encontra-se mais baixa do que a que representa a fração para conteúdos muito populares pelos mesmos motivos explicados na Figura 5.1.

Em contraste ao que foi observado na Figura 5.10, notamos um comportamento diferente ao observarmos a métrica ilustrada na Figura 5.11. O tempo médio de vida no cache possui um comportamento único para cada classe de popularidade. Para conteúdos muito populares, a média se manteve constante próxima ao valor de

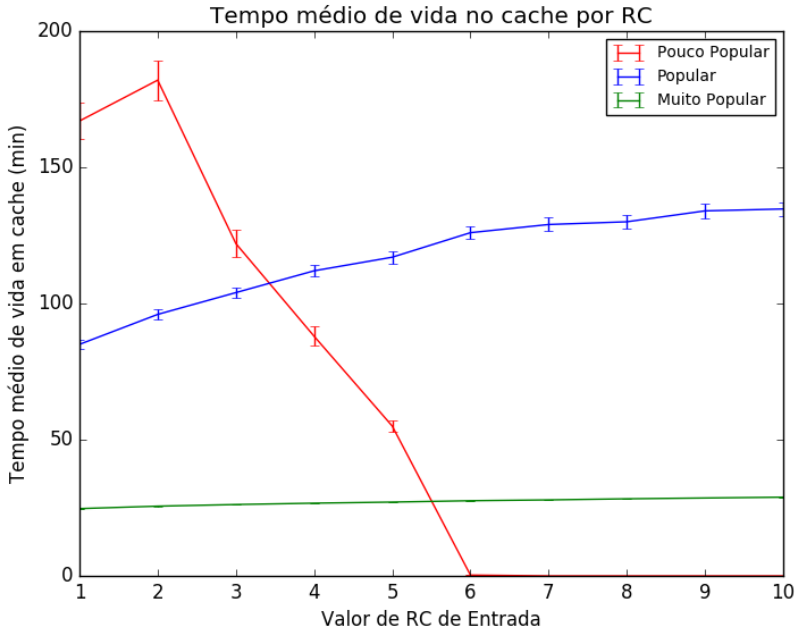

Figura 5.11: Tempo médio de vida no cache, por valor de RC.

$1/\mu$, conforme explicado pela Figura 5.5. Para conteúdos pouco populares, há um decréscimo linear até chegar a zero, quando nenhum conteúdo consegue entrar no cache. Porém, para conteúdos populares, notamos um leve crescimento conforme aumentamos o valor do RC de entrada, o que indica que há uma maior sobrevida no cache dos conteúdos populares que conseguem entrar no cache. Isso pode ser explicado por dois fatores: (1) os conteúdos pouco populares não entram no cache a partir de um certo valor do RC de entrada e isso diminui a competitividade pelo mesmo; (2) com o aumento do valor do RC de saída do cache, os conteúdos populares são mantidos no cache por mais tempo.

Observando, por fim, a métrica número de escritas no cache, vemos pela Figura 5.12 que há um decréscimo no número de escritas no cache conforme aumentamos o valor do RC de entrada. Esse resultado é esperado pois a medida que o valor do RC de entrada aumenta, os conteúdos pouco populares não conseguem entrar no cache, e aqueles que entram demoram mais para sair.

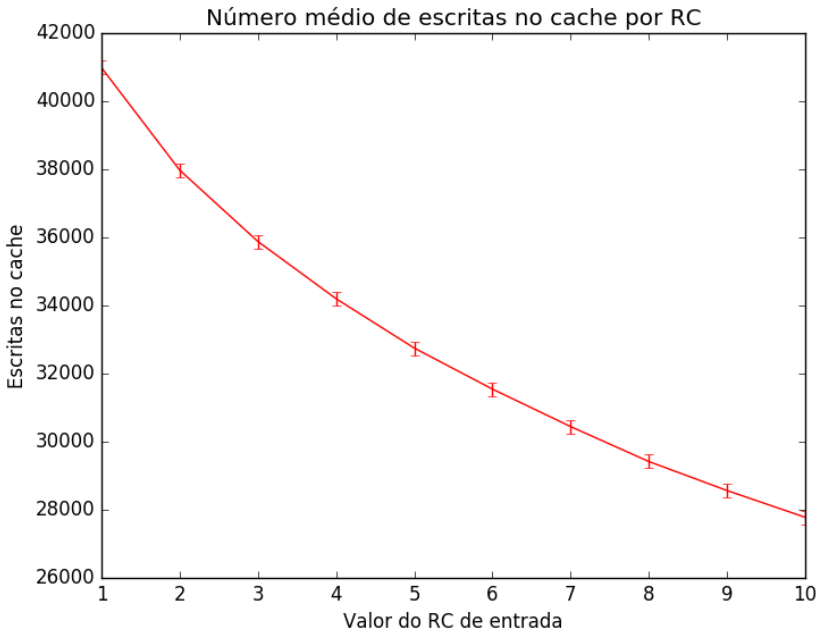

Figura 5.12: Número de escritas no cache, por valor de RC.

5.5 Histerese

Nesta seção, diferentemente da anterior, realizaremos experimentos fixando o valor de entrada do RC em 10, e variando o valor de saída de 0 a 9, variando portanto entre o máximo de histerese que podemos ter no sistema (diferença de valor 10) e não ter histerese alguma. Manteremos os demais parâmetros fixos nos seus valores padrão para impedir qualquer interferência na análise dos resultados.

Notamos na Figura 5.13 que a fração de requisições filtradas para conteúdos muito populares não sofreu uma grande alteração ao inserirmos a histerese no sistema. Houve uma pequena melhora com pouca histerese, seguida de um pequeno decréscimo ao aumentarmos a diferença. O mesmo ocorre para a fração total de requisições. Observando a escala do eixo Y da Figura 5.13 nota-se que a diferença numérica é bem pequena entre o menor e o maior valor de histerese. Para os conteúdos populares podemos observar na Figura 5.14 um pequeno aumento na fração de requisições filtradas com o aumento da histerese. Para esses conteúdos, o valor do parâmetro $1/\mu$ (intervalo para descréscimo do RC) é alto. Com a histerese máxima, levará muito tempo para o valor do RC de saída ser alcançado e o conteúdo ser removido do cache.

A Figura 5.15 mostra o tempo médio de vida no cache aumentando linearmente com a histerese, o que é de se esperar pois o número de decréscimos do RC necessários para sair do cache também cresce linearmente. Observamos que os conteúdos de popularidade muito elevada têm uma inclinação da reta menor que os de popularidade média, pois o valor de seu parâmetro $1/\mu$ (decréscimo do RC) é menor. Devido a isso

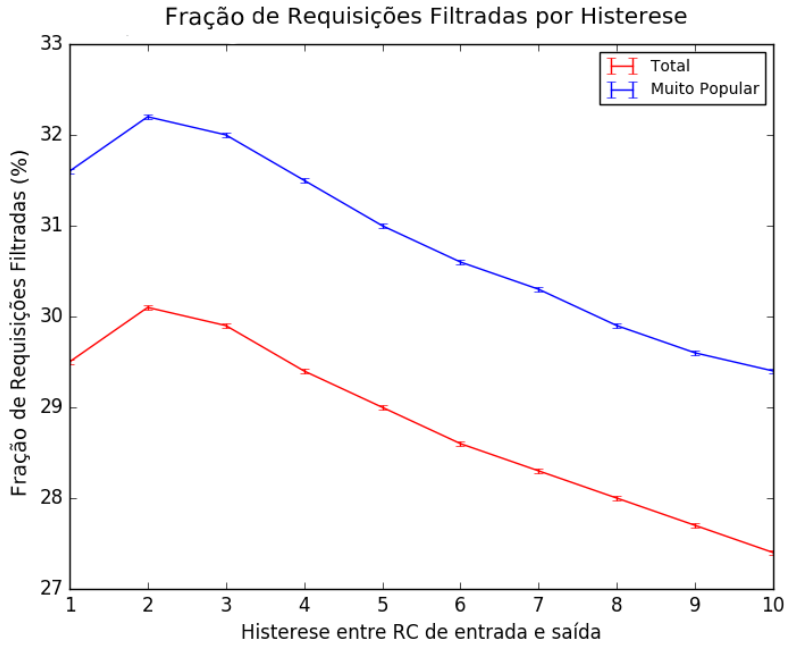

Figura 5.13: Fração de requisições filtradas, por diferença de RC.

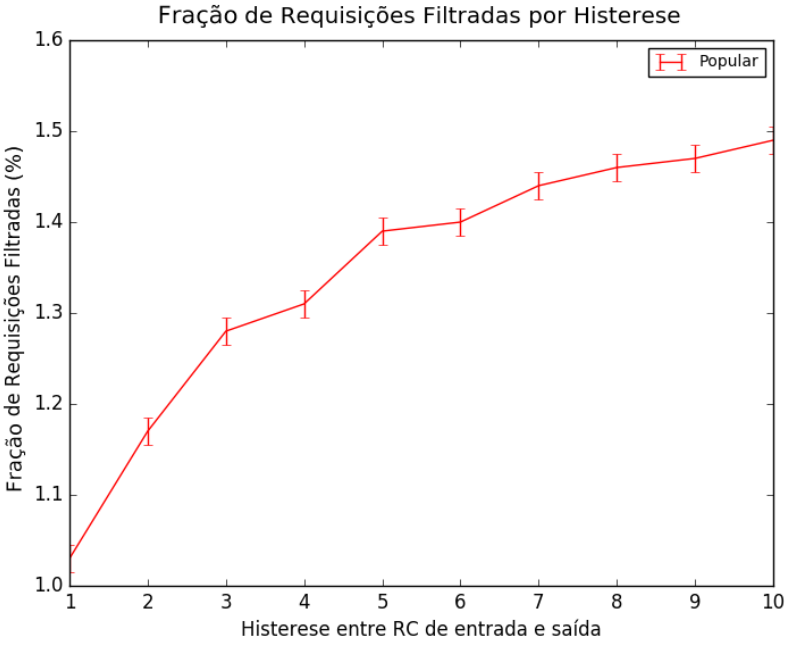

Figura 5.14: Fração de requisições filtradas para popularidade média, por diferença de RC.

e à maior competitividade pelo espaço no cache, a fração de requisições filtrada para conteúdos muito populares tende a cair um pouco conforme visto na Figura 5.13.

Por fim, observando a Figura 5.16 notamos que há decréscimo de uma ordem de grandeza no número de escritas no cache conforme aumentamos a histerese, pois os conteúdos que conseguem entrar têm um tempo de vida no cache maior do que sem

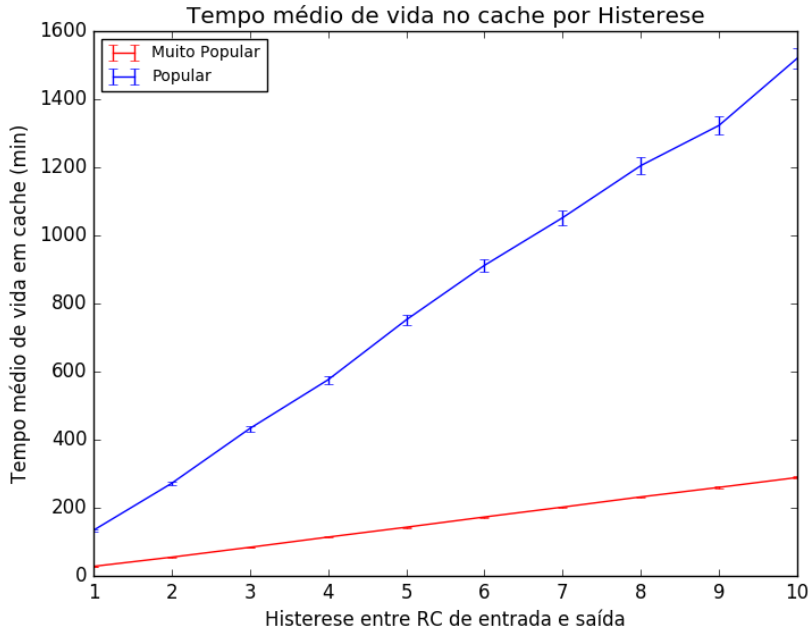

Figura 5.15: Tempo médio de vida no cache, por diferença de RC.

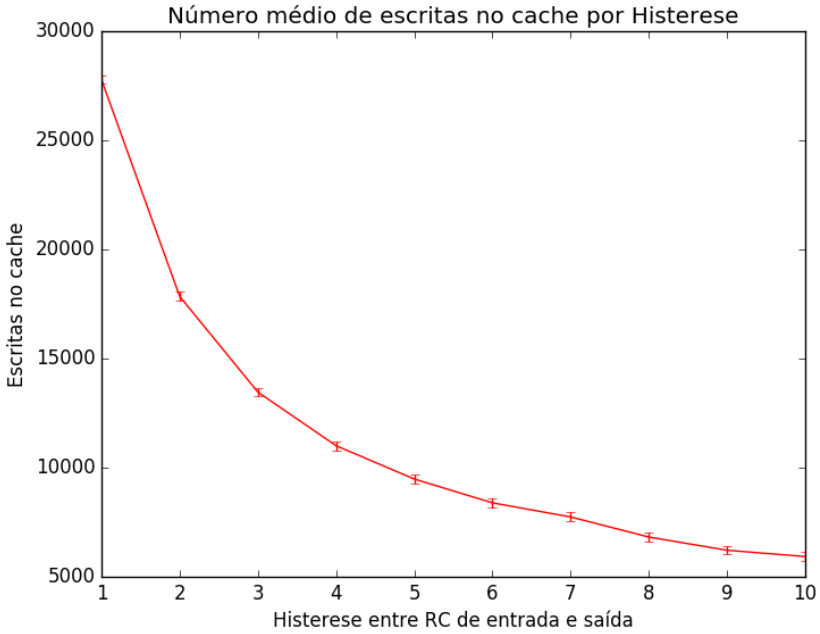

Figura 5.16: Número de escritas no cache, por diferença de RC.

histerese, conforme visto pela Figura 5.15.

Pelos resultados acima podemos concluir que a histerese mostra-se um mecanismo que favorece a diminuição do número de escritas no cache, pois conteúdos são mantidos por mais tempo no cache, sem diminuir significativamente a fração total de requisições fitradas.

5.6 Políticas de Troca no Cache

Para avaliar como cada política de remoção do cache afeta as métricas de interesse, não utilizaremos gráficos como nas seções anteriores, mas sim tabelas comparando diferentes políticas de cache para diferentes variações de parâmetros.

Primeiramente, vamos observar como a variação do tamanho de cache altera as métricas de fração de requisições filtradas e tempo médio de vida no cache, para cada classe de popularidade.

Cache 5	Cache 20	Cache 50
21.16 (RC)	32.33 (RC)	33.96 (LRU)
19.27 (LFU)	31.76 (LFU)	33.96 (LFU)
15.22 (LRU)	31.42 (LRU)	33.96 (FIFO)
15.11 (FIFO)	31.29 (FIFO)	33.93 (RC)

Tabela 5.1: Frações de requisições filtradas por cache.

Na Tabela 5.1, podemos ver que com um tamanho de cache médio ou alto, não há muita diferença na política de remoção utilizada, pelo fato de não haver muitas trocas de conteúdo quando comparado a um tamanho de cache pequeno. Para um cache pequeno, há uma clara vantagem das políticas RC e LFU sobre as políticas LRU e FIFO, o que pode ser explicado pelo fato de ambas serem sensíveis a rajadas de conteúdos, favorecendo conteúdos que são acessados muitas vezes em um espaço de tempo curto. Já a política LRU favorece conteúdos que são acessados de maneira uniforme no tempo, e a política FIFO favorece a ordem de chegada dos conteúdos do cache. Como os conteúdos na simulação são acessados em rajada mais frequentemente do que seguindo espaços uniformes de tempo, faz sentido as políticas que favoreçam esse tipo de acesso terem vantagem. Na Tabela 5.2 podemos ver que o mesmo comportamento é observado se isolarmos a métrica para conteúdos muito populares.

Cache 5	Cache 20	Cache 50
22.60 (RC)	34.08 (RC)	35.63 (FIFO)
20.52 (LFU)	33.44 (LFU)	35.62 (RC)
16.06 (LRU)	33.10 (LRU)	35.62 (LFU)
15.94 (FIFO)	32.96 (FIFO)	35.59 (LRU)

Tabela 5.2: Frações de requisições filtradas por cache para conteúdos muito populares.

Na Tabela 5.3 e na Tabela 5.4, observamos um comportamento diferente do visto acima. Para conteúdos de popularidade média e baixa as políticas LRU e FIFO apresentam melhor desempenho para tamanhos pequenos de cache. Isto ocorre pois as políticas LRU e FIFO não levam em conta o número de requisições que um

conteúdo recebeu na hora de removê-lo do cache. Dessa forma, os conteúdos de popularidade média e baixa tem maior chance de se manter no cache, aumentando a fração de requisições filtradas.

Cache 5	Cache 20	Cache 50
3.91 (LRU)	9.19 (LFU)	11.66 (RC)
3.85 (FIFO)	8.85 (LRU)	11.63 (LFU)
2.39 (LFU)	8.84 (RC)	11.60 (FIFO)
1.72 (RC)	8.82 (FIFO)	11.58 (LRU)

Tabela 5.3: Frações de requisições filtradas por cache para conteúdos populares.

Cache 5	Cache 20	Cache 50
0.14 (LRU)	0.32 (LFU)	0.44 (FIFO)
0.11 (FIFO)	0.24 (FIFO)	0.40 (RC)
0.08 (LFU)	0.22 (LRU)	0.36 (LRU)
0.07 (RC)	0.14 (RC)	0.36 (LFU)

Tabela 5.4: Frações de requisições filtradas por cache para conteúdos pouco populares.

Observando agora a métrica de tempo médio de vida no cache, para conteúdos de popularidade média e baixa, notamos pela Tabela 5.5 e pela Tabela 5.6 que as políticas LFU e RC mantém os conteúdos por mais tempo no cache. No entanto, observamos que ficar um tempo maior no cache não necessariamente implica em um aumento da fração de requisições filtradas para conteúdos de média/baixa popularidade. A chance de outra rajada de requisições por um desses conteúdos chegar antes dele ser trocado no cache, é bem baixa em condições de tamanho de cache limitado. Embora as políticas mais sensíveis a rajadas aumentem o tempo que o conteúdo fica no cache, é necessário que uma outra rajada chegue para consumir o conteúdo armazenado, para haver um aumento na fração de requisições filtradas. Se isto não ocorrer e uma rajada de algum outro conteúdo aparecer, esse conteúdo será retirado do cache.

Cache 5	Cache 20	Cache 50
12 (LFU)	**164 (LFU)**	290 (LFU)
8 (LRU)	**160 (RC)**	288 (LRU)
8 (FIFO)	105 (FIFO)	287 (RC)
5 (RC)	104 (LRU)	287 (FIFO)

Tabela 5.5: Tempo médio de vida no cache para conteúdos populares.

Mudando agora os parâmetros da simulação para o limiar do valor de RC ao invés do tamanho do cache, os resultados obtidos são bem similares, sendo justificados pelos mesmos motivos descritos na análise das tabelas acima. Os limiares utilizados

Cache 5	Cache 20	Cache 50
41 (LFU)	**1583 (LFU)**	**6580 (LFU)**
5 (LRU)	**482 (RC)**	**6390 (RC)**
4 (FIFO)	122 (LRU)	4613 (LRU)
1 (RC)	115 (FIFO)	4518 (FIFO)

Tabela 5.6: Tempo médio de vida no cache para conteúdos pouco populares.

foram 1, 3 e 10, e é importante notar que com o limiar mais alto os conteúdos muito pouco populares sequer conseguem entrar no cache.

RC 1	RC 3	RC 10
34.24 (RC)	34.08 (RC)	32.27 (RC)
33.11 (LFU)	33.44 (LFU)	31.93 (LFU)
33.09 (LRU)	33.10 (LRU)	31.67 (LRU)
32.91 (FIFO)	32.96 (FIFO)	31.54 (FIFO)

Tabela 5.7: Frações de requisições filtradas por RC para conteúdos muito populares.

RC 1	RC 3	RC 10
16.21 (FIFO)	9.19 (LFU)	1.29 (LFU)
16.18 (LRU)	8.85 (LRU)	1.28 (RC)
16.16 (LFU)	8.84 (RC)	1.03 (FIFO)
16.15 (RC)	8.82 (FIFO)	1.03 (LRU)

Tabela 5.8: Frações de requisições filtradas por RC para conteúdos populares.

Podemos ver pela Tabela 5.7 e pela Tabela 5.8 que o comportamento para conteúdos muito populares quase não sofre alterações, pois afinal não são muito afetados pela variação do limiar do valor do RC, conforme visto anteriormente. O mesmo pode ser dito para os conteúdos de popularidade média.

Olhando agora para os valores de tempo médio de vida no cache para conteúdos de popularidades média e baixa, conforme Tabela 5.9 e Tabela 5.10, observamos que as políticas LFU e RC são as que mais favorecem a permanência desses conteúdos no cache. O que está de acordo com os resultados obtidos quando variamos o tamanho do cache (Tabela 5.5 e Tabela 5.6).

RC 1	RC 3	RC 10
124 (RC)	**164 (LFU)**	**237 (LFU)**
123 (LFU)	**160 (RC)**	**225 (RC)**
85 (LRU)	106 (LRU)	136 (FIFO)
86 (FIFO)	105 (FIFO)	134 (LRU)

Tabela 5.9: Tempo médio de vida no cache por RC para conteúdos populares.

RC 1	RC 3	RC 10
1282 (LFU)	**1583 (LFU)**	0 (LFU)
709 (RC)	**482 (RC)**	0 (RC)
167 (LRU)	122 (LRU)	0 (LRU)
167 (FIFO)	115 (FIFO)	0 (FIFO)

Tabela 5.10: Tempo médio de vida no cache por RC para conteúdos muito pouco populares.

5.7 Políticas de Inserção e Remoção no Cache

Por fim, compararemos as duas políticas de inserção no cache implementadas, as políticas TTL e RC. Para tanto, repetimos os experimentos descritos na Seção 5.2 e na Seção 5.3 usando a política de inserção TTL, e então geramos gráficos com ambos os resultados para fazermos uma comparação entre as políticas. Dessa forma, podemos avaliar qualquer mudança de comportamento nas métricas, assim como diferenças numéricas.

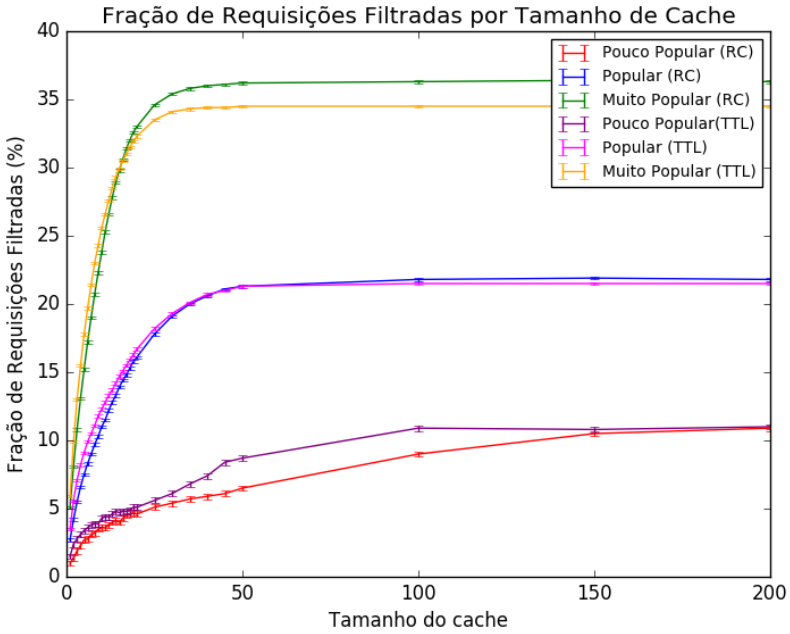

Figura 5.17: Fração de requisições filtradas, por tamanho de cache, para ambas políticas de inserção.

Primeiramente observamos a fração de requisições filtradas variando o tamanho de cache, para ambas as políticas na Figura 5.17. Pode-se notar que existe uma diferença no valor da métrica para conteúdos muito populares e pouco populares, quando o tamanho de cache é maior que 50. Conteúdos muito populares apresentaram desempenho melhor com a política RC, ao passo que conteúdos pouco populares tiveram um desempenho melhor com a política TTL. Como a política

TTL não mantém um contador de requisições, um conteúdo extremamente popular que permaneceria no cache usando a política RC, pode não permanecer no cache. Isso ocorre pois se uma requisição não chegar antes do tempo de expiração usado pela política TTL, o conteúdo é removido imediatamente do cache, ao passo que na política RC ele deveria esperar o valor de RC cair até o limiar de saída. Portanto a política TTL, favorece a saída de conteúdos muito populares e aumenta a chance de conteúdos pouco populares ocuparem o cache.

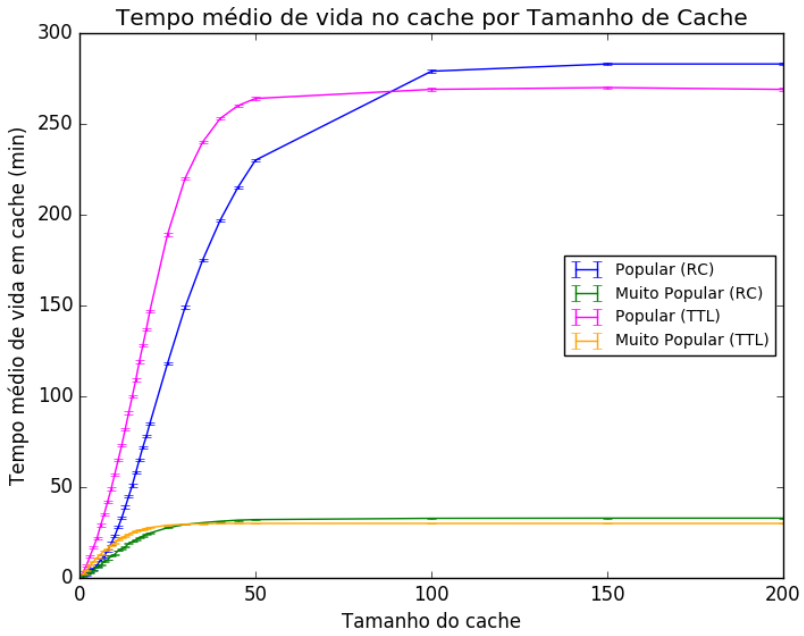

Figura 5.18: Tempo médio de vida no cache, por tamanho de cache, para ambas políticas de inserção.

Observando a Figura 5.18 notamos que não existe diferença significativa no valor do tempo médio de vida no cache para as políticas RC e TTL. Vemos que em ambos os casos os tempos convergem para valores próximos do parâmetro $1/\mu$. Já na Figura 5.19, notamos que há uma diferença significativa no número médio de escritas no cache. A política RC reduz o número de remoções e reinserções do mesmo conteúdo no cache por armazenar um contador de requisições, portanto apresenta um desempenho melhor que a política TTL.

Variando agora o número de saltos aleatórios, temos resultados similares aos observados anteriormente. As curvas apresentam o mesmo comportamento, indicando que a melhora vista pelo aumento de número de saltos aleatórios está presente independente da política de inserção usada. Novamente, podemos obeservar que a política TTL apresenta desempenho ligeiramente superior para conteúdos de popularidade média e baixa, ao passo que a política RC favorece conteúdos de popularidade alta, conforme explicado anteriormente.

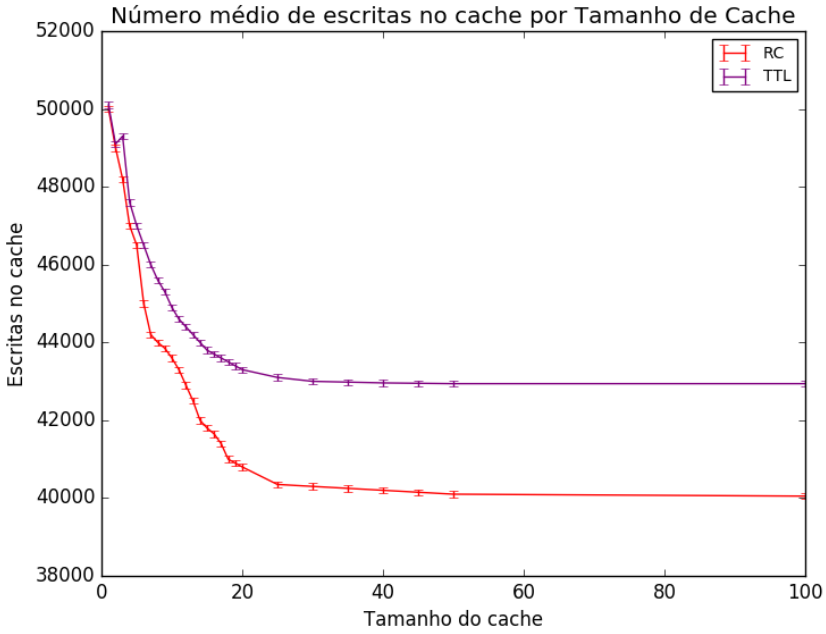

Figura 5.19: Número de escritas no cache, por tamanho de cache, para ambas políticas de inserção.

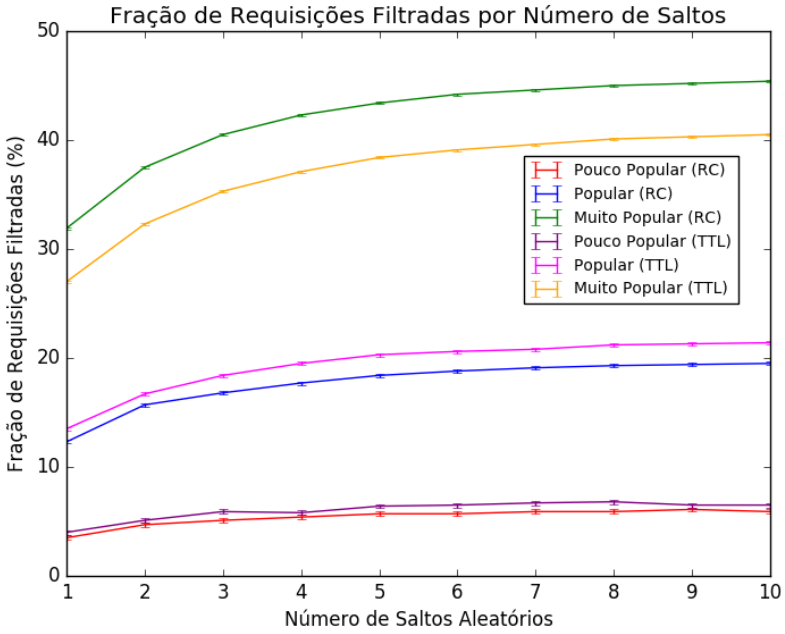

Figura 5.20: Fração de requisições filtradas, por número de saltos, para ambas políticas de inserção.

Esse favorecimento da política TTL para conteúdos menos populares fica claro na Figura 5.21, em que observamos um tempo médio de vida no cache consideravelmente maior para conteúdos populares na política TTL do que na política RC. Lembramos que com o tamanho de cache utilizado, esse tempo médio ainda não

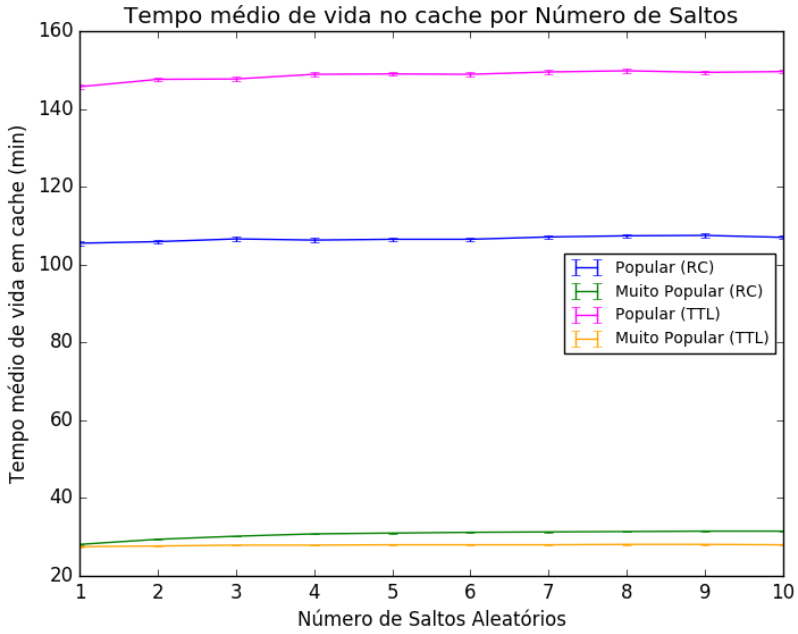

Figura 5.21: Tempo médio de vida no cache para conteúdos muito populares, por número de saltos, para ambas políticas de inserção.

converge para o valor de $1/\mu$, e que o tempo médio não se altera ao aumentarmos o número de saltos aleatórios pelos mesmos motivos explicados na Figura 5.9.

Por fim, a mesma vantagem numérica da política RC sobre a TTL observada no número médio de escritas no cache ao variarmos o tamanho de cache foi observada ao variarmos o número de saltos aleatórios, como podemos ver na33 Figura 5.22.

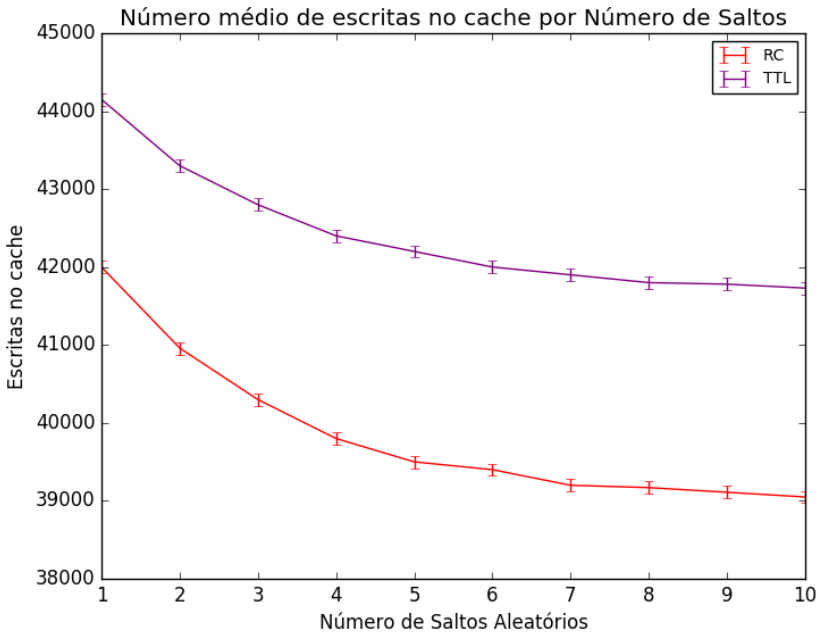

Figura 5.22: Número de escritas no cache, por número de saltos, para ambas políticas de inserção.

Capítulo 6

Conclusões

6.1 Considerações Finais

Nesta monografia foi desenvolvido um simulador de redes de cache cujo objetivo principal foi avaliar o desempenho da arquitetura proposta em [1], usando dados reais de uma empresa provedora de serviços de vídeo on demand. Para chegar aos resultados apresentados foi desenvolvido um método de pré-processamento adequado aos dados utilizados, cujos resultados foram usados para parametrizar o simulador e os caches da topologia.

Dentre os vários resultados apresentados no Capítulo 5, destacamos abaixo os que apresentaram maior relevância no contexto de analisar o desempenho da rede:

- Aumentar o tamanho do cache resultou em um aumento da fração de requisições filtradas pela rede e no tempo médio de vida no cache, além da redução do número de escritas no cache. Todos os resultados foram favoráveis, porém foi notado que aumentar demasiadamente o tamanho do cache não traz muitos benefícios, pois a partir de certo tamanho as métricas se estabilizaram em certos valores.

- O tempo médio de vida no cache tende a se igualar à média da variável aleatória exponencial que governa o tempo de decréscimo do valor de RC, pelo fato de muitos conteúdos não conseguirem se manter no cache por falta de requisições.

- Aumentar o número de saltos do passeio aleatório trouxe benefícios tanto para a fração de requisições filtradas pela rede quanto para o número de escritas no cache, porém aumentar esse valor exageradamente não trouxe muitos benefícios, pois a partir de certo valor é muito improvável que algum conteúdo que esteja em algum cache não seja encontrado pelo passeio aleatório.

- Aumentar o valor de entrada e saída do RC não foi benéfico para as métricas observadas, à exceção do número de escritas no cache, pois a fração de

conteúdos que consegue entrar no cache é reduzida conforme aumentamos o limiar do RC. Notamos porém que os conteúdos populares que de fato conseguem entrar desfrutam de uma menor competição pelo espaço do cache.

- A inserção da histerese não trouxe um impacto forte na fração de requisições filtradas, piorando levemente o desempenho conforme aumentamos a histerese. Porém aumentamos o tempo médio de vida no cache, pois este é uma função linear do número de decréscimos ao RC necessários para sair do cache. Diminuímos consideravelmente o número médio de escritas no cache, em função desse maior tempo de vida.

- O impacto das políticas de troca e remoção no cache foi bem pequeno para a maior parte dos casos, houve um breve favorecimento das políticas LFU e RC por conta da sua sensibilidade a rajadas de requisições, que ocorrem frequentemente com os dados utilizados.

- A política de inserção no cache TTL ofereceu um desempenho melhor para conteúdos menos populares, ao passo que a política RC favoreceu os conteúdos mais populares.

6.2 Trabalhos Futuros

Alguns tópicos abordados neste trabalho não foram discutidos a fundo, e poderiam ser abordados com maior profundidade em algum trabalho futuro baseado neste. Além disso, podemos destacar outras abordagens que não foram mencionadas ao longo do trabalho, que possam expandir o escopo da pesquisa e aproximar o simulador de algo mais real. Abaixo temos alguns desses tópicos que podem ser abordados em trabalhos futuros:

- Comparar o impacto da presença de uma política de troca em cache com a ausência dela, conforme previsto no modelo em [1]. O modelo não prevê o uso de políticas de troca em cache pelo fato do próprio RC remover o conteúdo do cache. Um estudo poderia ser feito para comprovar (ou descartar) a necessidade de uma política de troca.

- Avaliar como o número de classes de popularidade influencia o desempenho da rede de cache, além de utilizar um procedimento mais rigoroso para chegar ao valor do parâmetro para taxa média de chegada de requisições (λ).

- Avaliar como a rede se comporta levando em conta os atrasos de comunicação entre nós da topologia, assim como considerar que os nós intermediários da rede não são servidores infinitos, mas sim servidores que atendem ao máximo

N requisições distintas, criando uma fila de eventos a serem atendidos em cada nó.

- Avaliar como outras topologias menos idealizadas alteram o desempenho da rede, ou ainda topologias maiores que aumentarão a quantidade de caches disponíveis. Alterar o grafo completo de cada camada por um látice regular com mutações nas arestas (modelo small world) pode ser um bom começo para avaliar diferenças no desempenho quando não se assume que o grafo é completo.

- Avaliar o caso em que conteúdos têm tamanhos diferentes, usando de políticas de inserção e troca de cache mais complexas para avaliar se vale a pena inserir um conteúdo grande no cache ou não, de acordo com sua popularidade.

Referências Bibliográficas

[1] DOMINGUES, G. "Redes orientadas à informação com estratégias probabilísticas acopladas para busca e replicação", . Tese de doutorado de Guilherme Domingues., 2015.

[2] KUROSE, J. "Information-centric networking: The evolution from circuits to packets to content", *Computer Networks*, v. 66, pp. 112 – 120, 2014. ISSN: 1389-1286. doi: https://doi.org/10.1016/j.comnet.2014.04.002. Disponível em: ¡http://www.sciencedirect.com/science/article/pii/S1389128614001455¿. Leonard Kleinrock Tribute Issue: A Collection of Papers by his Students.

[3] DOMINGUES, G., DE SOUZA E SILVA, E., LEÃO, R. M., et al. "Enabling opportunistic search and placement in cache networks", *Computer Networks*, v. 119, pp. 17 – 34, 2017. ISSN: 1389-1286. doi: https://doi.org/10.1016/j.comnet.2017.03.005. Disponível em: ¡http://www.sciencedirect.com/science/article/pii/S1389128617300737¿.

[4] XYLOMENOS, G., VERVERIDIS, C. N., SIRIS, V. A., et al. "A Survey of Information-Centric Networking Research", *IEEE Communications Surveys Tutorials*, v. 16, n. 2, pp. 1024–1049, Second 2014. ISSN: 1553-877X. doi: 10.1109/SURV.2013.070813.00063.

[5] FOFACK, N. C., NAIN, P., NEGLIA, G., et al. "Performance evaluation of hierarchical TTL-based cache networks", *Computer Networks*, v. 65, pp. 212 – 231, 2014. ISSN: 1389-1286. doi: https://doi.org/10.1016/j.comnet.2014.03.006. Disponível em: ¡http://www.sciencedirect.com/science/article/pii/S1389128614001108¿.

[6] MENDONÇA, G., DOMINGUES, G., SILVA, E., et al. "Escalando caching em redes orientadas a conteúdo via mecanismos de histerese", . Artigo submetido ao SBRC 2018., mar 2018.

[7] SQLITE. "SQLite - SQL Database Engine". . Disponível em: ¡https://www.sqlite.org/index.html¿.

[8] KHALEGHI, A., RYABKO, D., MARY, J., et al. “Consistent Algorithms for Clustering Time Series”, *Journal of Machine Learning Research*, v. 17, n. 3, pp. 1–32, 2016. Disponível em: ¡http://jmlr.org/papers/v17/khaleghi16a.html¿.

[9] MONTERO, P., VILAR, J. “TSclust: An R Package for Time Series Clustering”, *Journal of Statistical Software, Articles*, v. 62, n. 1, pp. 1–43, 2014. ISSN: 1548-7660. doi: 10.18637/jss.v062.i01. Disponível em: ¡https://www.jstatsoft.org/v062/i01¿.

[10] JSONCPP. “JsonCpp - C++ json library”. . Disponível em: ¡https://github.com/open-source-parsers/jsoncpp¿.

[11] STL. “STL - Standard Template Library for C++”. . Disponível em: ¡http://www.cplusplus.com/reference/stl/¿.